MSX
わかる
マシン語入門
新星出版社
309

わかる MSX マシン語入門

マシン語の一般知識とプログラミング

田中一郎
大沢昭二



新星出版社

MSX
シリーズ

はじめてのパソコン入門　山下利秋著

絵でわかる たのしいパソコン入門　関口泰夫著

よくわかる ぼくらのパソコン入門　山下利秋著

はじめてのプログラミング　大沢昭二・田中一郎著

わかる MSX マシン語入門

マシン語の一般知識とプログラミング

田中一郎
大沢昭二

新星出版社

はじめに

今日、ＭＳＸパソコンでも、マシン語が人気を集めています。

マシン語が人気を集める理由は、ＭＳＸベーシックとは比較にならないほど実行速度が速いからでしょう。

特にゲームプログラムでは、実行速度が速いというマシン語の特徴が素晴しい力を発揮しますから、自分なりに面白いゲームプログラムを作ってみたいと思う人には、ぜひ、おぼえてもらいたいプログラミング言語です。

ところで、マシン語はむずかしいという人が多くいますが、決してそんなことはありません。最初、ＭＳＸパソコンを買ってきて、ＭＳＸベーシックにとりかかったときは、ＭＳＸベーシックさえもむずかしく感じたことでしょう。それとおなじで、最初はマシン語もむずかしく感じるかも知れませんが、マシン語を使っているうちにそうではなくなります。つまり、マシン語に慣れることです。

この本は、これからマシン語を学んでみようという人のために、また、マシン語を学びはじめてまもない人のために書きました。したがって、マシン語でプログラムを作るときに必要な事柄や、マシン語のプログラムの書き表わし方、マシン語の命令などについて、簡単なプログラムを使って具体的にやさしく説明しました。

特に目でみて、マシン語のプログラムの働きが理解できるように、テキスト画面やスプライト面にキャラクタを表示したり、キャラクタを移動させたりするプログラムを多く取り上げていますから、だれもが楽しみながらマシン語をおぼえていくことができると思います。

この本がマシン語をおぼえるために、多くの人たちのお役にたてば、大変嬉しく思います。

田中一郎／大沢昭二

もくじ

マシン語はどんな形をしているか

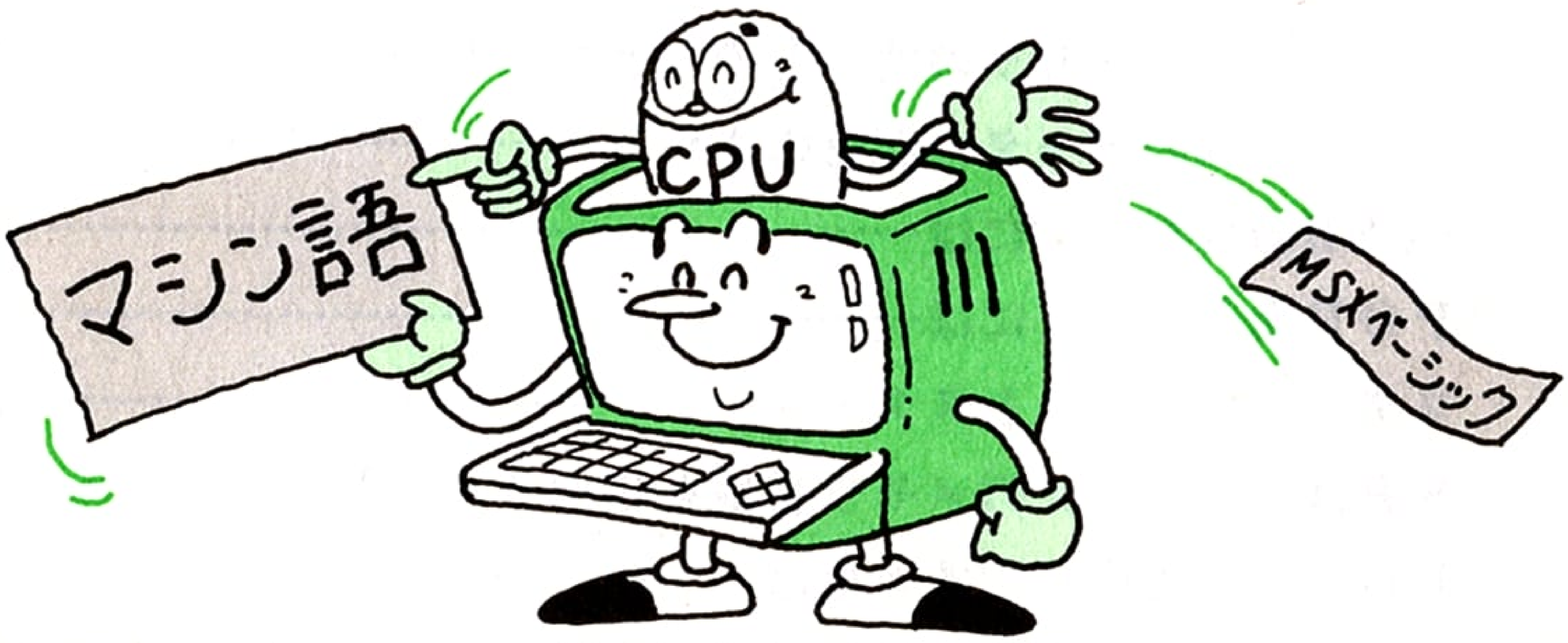
CPUは、マシン語しかわからない

ＭＳＸパソコンの内部に、ＣＰＵ（Central Processing Unit・中央演算処理装置）というものがあるということは知っていると思います。

ＣＰＵは、ＭＳＸパソコンの頭脳といわれる部分です。ＣＰＵがＭＳＸパソコンの頭脳といわれるのは、プログラムの命令が、どういう意味の命令であるかを解読したり、その命令の意味にしたがっていま何を行うかをＭＳＸパソコンの各部に知らせたり、また、計算や判断などを行ったりするからです。

ところで、いま説明したように、ＣＰＵはプログラムの命令がどういう意味の命令であるか解読しますが、どんなプログラミング言語で書かれた命令でも解読できるのかというと、そうではありません。

ＭＳＸパソコンのプログラミング言語には、ＭＳＸベーシックとマシン語のふたつがありますが、ＣＰＵが解読できる命令はマシン語の命令だけで、ＭＳＸベーシックの命令は、解読することができません。



では、ＣＰＵがＭＳＸベーシックの命令を解読することができないのに、なぜＭＳＸベーシックで書かれたプログラムが実行されるのでしょうか。その理由は、ＭＳＸパソコンのなかには、ベーシックインタプリタという、ＭＳＸベーシックをマシン語に翻訳(ほんやく)するプログラムが内蔵されていて、ＣＰＵが命令の意味を解読できるように、ＭＳＸベーシックの命令をひとつひとつマシン語に翻訳しているからです。

ＭＳＸベーシックの実行速度がいやになるほど遅いのは、このようにベーシックインタプリタでマシン語に変換するという、余計な仕事が加わるからです。これに対してマシン語は、直接ＣＰＵに解読されるので、それだけ実行速度が速いのです。

マシン語はどんな形？

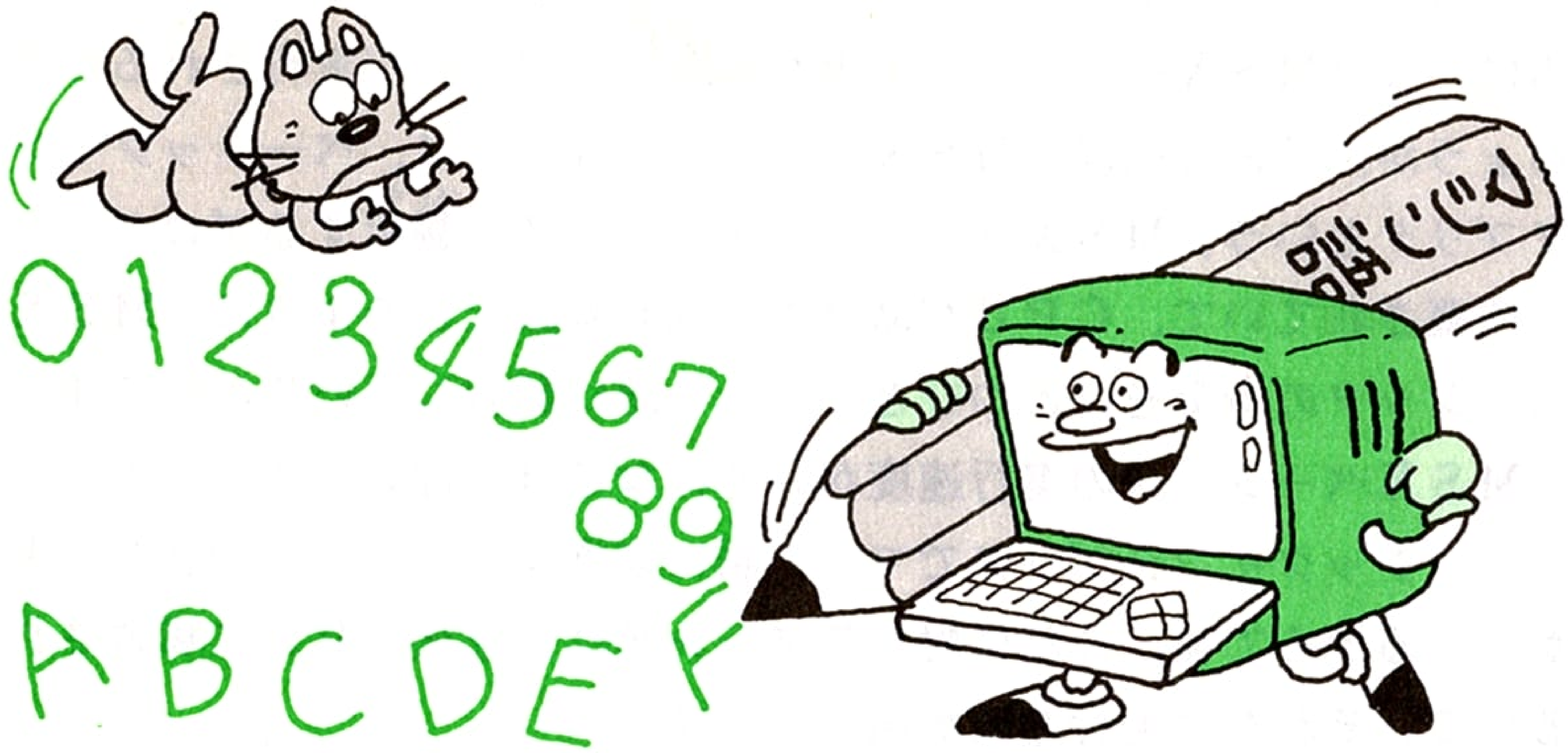
では、ＣＰＵで解読することができる唯一の言葉、マシン語とは、いったいどんな形をしているのでしょうか。ここではマシン語の形を、マシン語のプログラムで探ってみることにします。

E000	3E	E008	DD
E001	0D	E009	F3
E002	32	E00A	3E
E003	DC	E00B	85
E004	F3	E00C	CD
E005	3E	E00D	A2
E006	10	E00E	00
E007	32	E00F	C9

これがマシン語のプログラムです。このプログラムがどんなプログラムであるかは、19頁で説明します。

マシン語のプログラムを見ると、数字とアルファベットなどの４桁で書き表わされているものと、２桁で書き表わされているものの２種類があります。

数字とアルファベットなどの４桁で書き表わされているものが、



E000　　E00A　　E00F ←メモリ番地

メモリ番地です。メモリ番地は、アドレスともいいます。

数字やアルファベットなどの２桁で書き表わされているものが、

3E　0E　DC　32　DD ←マシン語

マシン語です。

メモリ番地は４桁、マシン語は２桁ですが、ともに数字とアルファベットで書き表わされています。数字は、

0　1　2　3　4　5　6　7　8　9

０から９まで使われています。また、アルファベットは、

A　B　C　D　E　F

ＡからＦまでが使われています。

０から９までが数であることは、だれが見てもわかりますが、ＡからＦまでのアルファベットは、なにを表わしているのでしょうか。これも数を表わしています。ＡからＦまでのアルファベットで表わしている数は、私たちがふだん使っている10進数の、10から15までを表わしています。

A　B　C　D　E　F ←16進数

↕　↕　↕　↕　↕　↕

10　11　12　13　14　15 ←10進数

数字とアルファベットで表わす数とは

さてマシン語は、0から9までの数字とAからFまでのアルファベットの2桁で表わすということがわかりました。またAからFまでがどんな数を表わしているかもわかりました。このように0から9までとAからFまでの、16個の数字とアルファベットで表わす数のことを**16進数**といいます。

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	←16進数
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	←10進数

16進数は、10進数が、10を一区切りにして数を表わしているように、16を一区切りにしています。つまり、10進数は、0から9までが1桁ですが、16進数は、0からF（10進数の15）までが1桁になります。

ところで、16進数で10から15までをAからFで表わす理由は、10、11、12、13、14、15は、10進数では2桁の数になっていますから、これをそのまま、16進数で1桁の数として使うことができないからです。したがって、10から15までを1桁で表わすために、新しい数字を作る必要があります。新しい数字は何でもいいのですが、16進数ではA、B、C、D、E、Fを使って、10、11、12、13、14、15を、1桁の数として表わすようにしているわけです。

このため、10進数は0から9まで数えてきて、9のつぎで桁上がりして10と2桁になりますが、

10進数＝9のつぎで10と桁上がりして2桁になる

16進数では、0からF（10進数の15）まで数えてきて、Fのつぎで10と桁上がりして2桁になります。

16進数＝F（10進数の15）のつぎで10と桁上がりして2桁になる

このように16進数では、Fのつぎで10と桁上がりして2桁になりますから、10は数字通りの意味ではなくて10進数の16の数に相当します。

16進数の10＝10進数の16



このような16進数という数の表わし方ができたのは、２進数を使ってマシン語を書き表わすことがとても面倒だからです。つぎに13頁に示したマシン語のプログラムを、２進数で示します。

00111110	11011101
00001101	11110011
00110010	00111110
11011100	10000101
11110011	11001101
00111110	10100010
00010000	00000000
00110010	11001001

このように２進数の並び、つまり、１と０の８個の並びでマシン語を書き表わしていたのでは、目がチカチカして書き間違えたり、見間違えたりして、大変な作業になることはわかるでしょう。そこで16進数という数を作って、マシン語を書き表わすようにしたのです。

これで、マシン語の形がわかりました。

マシン語＝16進数の２桁で書き表わすもの

つまりマシン語は、16進数の２桁で書き表わすということです。

アセンブリ言語

アセンブリ言語とは

12頁で、マシン語の形を探りました。そしてマシン語は、16進数でできていることがわかりました。もちろんMSXパソコンのなかで、入力されたマシン語の16進数がそのままの形で記憶されて、やりとりされるわけではありません。

キーボードから入力されたマシン語の16進数は、入力されるとともに1と0でできた2進数に変換されて、MSXパソコンの本体のなかに入っていきますから、1と0でできた2進数で記憶されて、やりとりされることになります。もちろん、1と0がひとつひとつで記憶されて、やりとりされるというわけではありません。

MSXパソコンは、8ビット型のパソコンですから、

```
00111110 ←——— 3E
00001101 ←——— 0D
00110010 ←——— 32
```

というように、1と0の8個の並びで記憶して、8個の並びをひとつのまとまりとしてやりとりします。

1あるいは0のひとつひとつを、**1ビット**といいます。したがって1と0の8個の並びは、**8ビット**になります。

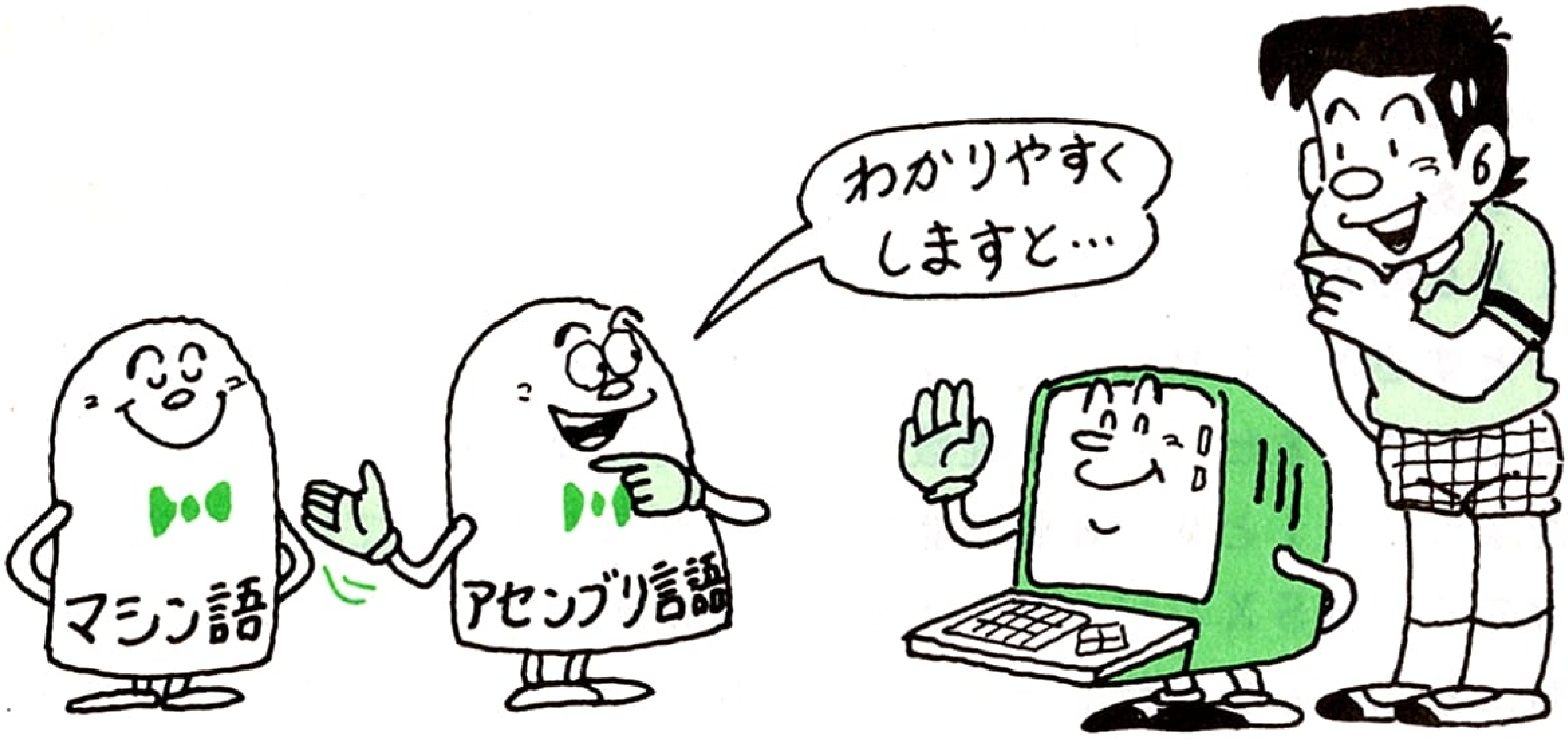
とすると、CPUが解読することができるマシン語とは、16進数ではなくて、8ビットの2進数ということになりますが、16頁で説明したように、1と0の8個の並びでプログラムを書いていたのでは大変なので、MSXパソコンが2進数に変換するのに便利であって、私たちにとっても2進数よりまだましな16進数で、マシン語のプログラムを書き表わすようになったわけです。

アセンブリ言語は、マシン語をわかりやすくする

ところでマシン語を16進数で書き表わしたからといって、はたして私たちにとって、マシン語がわかりやすくなったといえるでしょうか。

メモリ番地	マシン語	メモリ番地	マシン語
B000	3E 0D	B00A	3E 85
B002	32 DC F3	B00C	CD A2 00
B005	3E 10	B00F	C9
B007	32 DD F3		

たしかに、1と0の8個の並びの2進数で書くよりも、16進数のほうがプログラムを書くのが楽だ、ぐらいのことはいえそうです。しかし16進数といってもやはり数字ですから、私たちにとってわかりにくいものであることに変わりありません。なぜなら、16進数の数字を見ただけでは、マシン語の命令の意味はわかりませんし、16進数の数字で、マシン語の命令をおぼえることも大変だからです。そこで命令の意味を簡単な英文字の略記号で表わして、マシン語の命令のひとつひとつに対応させることにしました。それが**アセンブリ言語**です。



ではつぎに、まえの頁に示したマシン語のプログラムをアセンブリ言語で書き表わして、アセンブリ言語とはどんなものか見てみることにします。

メモリ番地	マシン語	アセンブリ言語
B000	3E　0D	LD　A，0DH
B002	32　DC　F3	LD　（F3DCH），A
B005	3E　10	LD　A，10H
B007	32　DD　F3	LD　（F3DDH），A
B00A	3E　85	LD　A，85H
B00C	CD　A2　00	CALL　00A2H
B00F	C9	RET

真中がマシン語です。右側が、そのマシン語をアセンブリ言語で書き表わしたものです。

LDはLOADの略で、「～に入れよ」という命令

1行目の3E　0Dのマシン語を、アセンブリ言語で書き表わすと、

3E　0D　➡　LD　A，0DH

LD　A，0DHとなります。3Eのアセンブリ言語は、

3E　➡　LD　A

LD　Aです。このうちのLDが命令です。LD命令は、英語のLoad(ロード）を略したもので、ロード命令といいます。

LD＝Load（ロード）の略

英語のLoadは、「～に入れる」という意味です。LD命令のあとのAは、AレジスタのAです。

A＝Aレジスタ

したがってLD　Aは、Aレジスタに～を入れよということになります。

LD　A＝Aレジスタに～を入れよ

さて、Aレジスタになにを入れるのでしょうか。この場合は、LD

Aのあとに，で区切られてあるＯＤＨです。ＯＤのあとにはＨがついていますから、ＯＤＨは16進数の数値ということになります。

ＯＤＨ＝数値（10進数で13）

ＯＤＨは、10進数で13です。このようにアセンブリ言語で書かれたプログラムのなかで、16進数にＨがついているときは、その16進数は数値であることを示しています。数値には、わかりやすくいうと、データとメモリ番地とがあります。

Ｈがついているもの＝数値（データ、メモリ番地）を表わす

さて、これでマシン語の、

３Ｅ　ＯＤ

を見ただけではなんのことかわからなくても、アセンブリ言語の、

ＬＤ　Ａ，ＯＤＨ←ODHをAレジスタに入れよ

であると、数値のＯＤＨ（10進数で13）を、Ａレジスタに入れよということなんだな、ということがわかります。

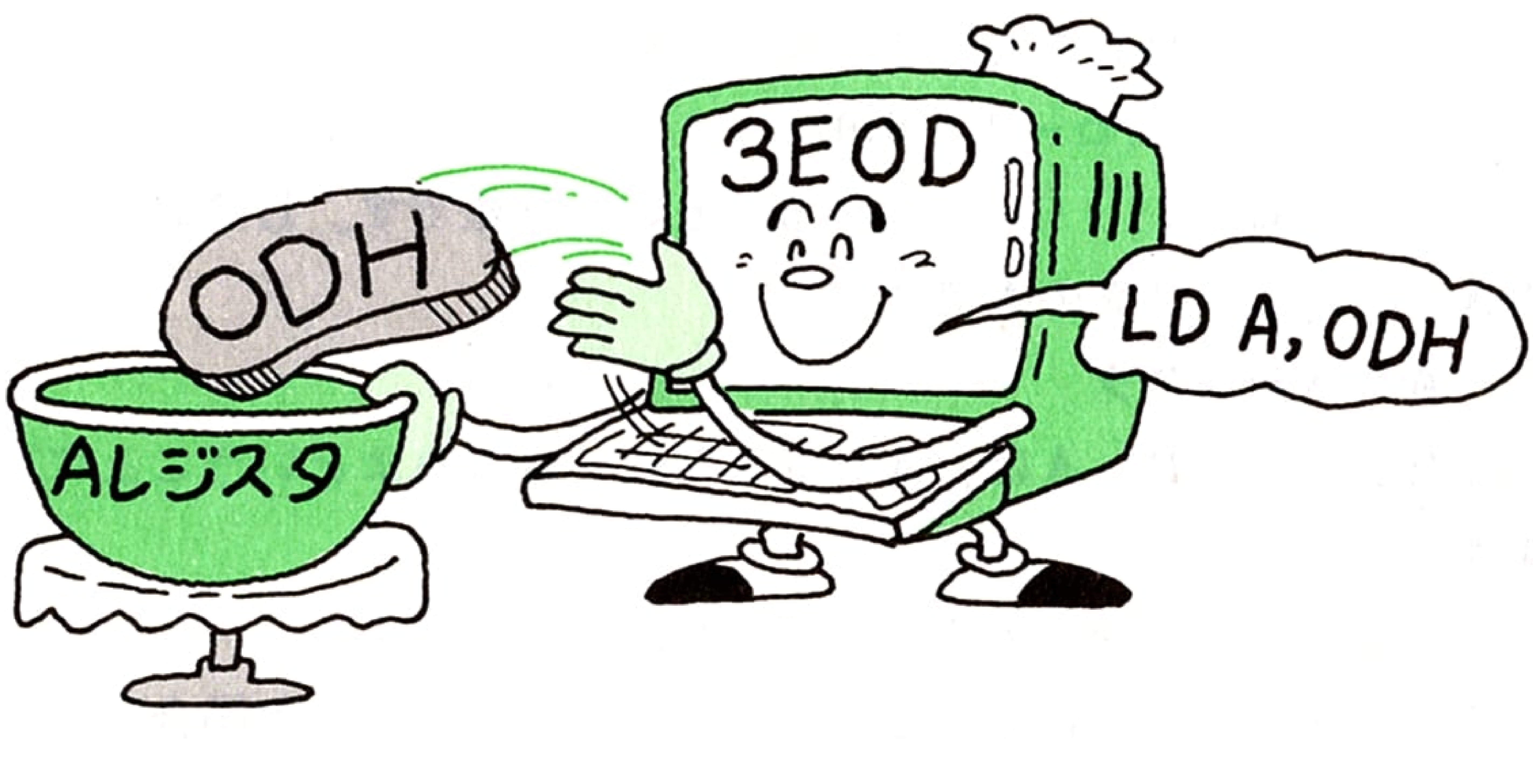
このように、私たちにとってわけのわからないマシン語を、私たちにわかるようにしてくれるのが、アセンブリ言語なのです。

では、２行目以降も説明しておくことにしましょう。

２行目の３２　ＤＣ　Ｆ３をアセンブリ言語で表わすと、

３２　ＤＣ　Ｆ３　➡　ＬＤ　（Ｆ３ＤＣＨ），Ａ

ＬＤ　（Ｆ３ＤＣＨ），Ａとなります。ＬＤは、～を入れよという



命令です。この場合は、ＬＤ命令のあとにカッコで括ってＦ３ＤＣＨがあります。

（Ｆ３ＤＣＨ）＝Ｈがついているので数値

Ｆ３ＤＣＨは、Ｈがついているので数値です。ただしＦ３ＤＣＨが数値であるといっても、１行目の０ＤＨと違ってメモリ番地です。ＬＤ命令のあとにカッコで括られて４桁の数値があるときは、その数値はメモリ番地と決まっています。

（Ｆ３ＤＣＨ）＝Ｆ３ＤＣＨはメモリ番地

そのあとにコンマで区切ってあるＡは、Ａレジスタのことです。したがって、ＬＤ　（Ｆ３ＤＣＨ），Ａは、Ａレジスタをカッコで括られたＦ３ＤＣ番地のメモリに入れよ、ということになります。もちろんＡレジスタをメモリに入れるなどということはできるはずがありません。つまり、Ａレジスタの内容（Ａレジスタに記憶されているもの）を、Ｆ３ＤＣ番地のメモリに入れよ、ということです。

ＬＤ　（Ｆ３ＤＣＨ），Ａ←Ａレジスタの内容をＦ３ＤＣ番地のメモリに入れよ

この場合は、１行目でＡレジスタに０ＤＨを入れていますから、この０ＤＨを、Ｆ３ＤＣ番地のメモリに入れよ、ということになります。

Aレジスタの内容0DH➡F3DC番地のメモリ



3行目の3E　10をアセンブリ言語で表わすと、

3E　10　➡　LD　A，10H

となります。LD　A，10Hは1行目とおなじです。Aレジスタに入れる数値が、10Hになっているにすぎません。したがって10H（10進数で16）をAレジスタに入れよ、ということになります。

LD　A，10H←10H（10進数で16）をAレジスタに入れよ

4行目の32　DD　F3　をアセンブリ言語で表わすと、

32　DD　F3　➡　LD　（F3DDH），A

となります。LD　（F3DDH），Aは、2行目とおなじです。Aレジスタの内容を入れるメモリの番地がF3DDHになっているにすぎません。したがって、F3DD番地のメモリにAレジスタの内容を入れよ、ということになります。

LD　（F3DDH），A←Aレジスタの内容をF3DD番地のメモリに入れよ

この場合は、3行目でAレジスタには10H（10進数の16）を入れましたから、F3DD番地のメモリには、10Hが入れられることになります。

5行目の3E　85をアセンブリ言語で表わすと、

3E　85　➡　LD　A，85H

となります。これは、1行目と3行目とおなじで、Aレジスタに85Hを入れよ、ということです。この85Hは、数値に違いありませんが、1行目と3行目でAレジスタに入れた数値と違って、●のキャラクタコードです。

1行目でAレジスタに入れて、F3DC番地のメモリに記憶させた0DHは●を表示させるための行座標となる数値です。3行目でAレジスタに入れて、F3DD番地のメモリに記憶させた10Hは●を表示させる列座標となる数値です。

1行目の0DH＝●を表示させる行座標

2行目の10H＝●を表示させる列座標

3行目の85H＝●のキャラクタコード

このように数値にもいろいろあるので、注意しなければなりません。

CALLは、サブルーチンにジャンプさせる命令

6行目のＣＤ　Ａ２　００をアセンブリ言語で表わすと、

CD　A2　00　➡　CALL　00A2H

となります。ＣＤのアセンブリ言語がＣＡＬＬです。

CD　➡　CALL

ＣＡＬＬは英語のＣＡＬＬ（コール）そのままで、**コール命令**といいます。英語のＣＡＬＬは、「呼ぶ」という意味ですが、マシン語のＣＡＬＬ命令は、**サブルーチン・コール命令**といって、サブルーチンを呼ぶという明確な目的をもっています。つまり、サブルーチンにジャンプさせるときに使う命令ということです。その意味ではベーシックのＧＯＳＵＢ命令とおなじといえます。

CALL命令＝サブルーチンにジャンプさせるときに使う命令

したがって、ＣＡＬＬ命令のあとには、サブルーチンの最初のメモリ番地を置きます。

CALL　<サブルーチンの最初のメモリ番地>

この場合はＣＡＬＬ　００Ａ２Ｈですから、

CALL　00A2H←00A2番地からのサブルーチンにジャンプせよ

００Ａ２番地からのサブルーチンにジャンプせよ、ということになります。

ところで、００Ａ２番地のメモリはどこにあるのかというと、ＲＯＭ領域にあるのです。ＲＯＭ領域には、いろいろな働きをするマシン語のプログラムが内蔵されていて、それぞれの働きをするマシン語のプログラムのひとつひとつを**ルーチン**といいます。そして、このルーチンのうちの入出力に関するルーチンの集まりを、**ＢＩＯＳ（バイオス）**といいます。ＢＩＯＳについては、154頁を参照してください。

このようにマシン語では、ＲＯＭのなかに内蔵されているルーチンをＣＡＬＬ命令で、サブルーチンとして使うことができるのです。

ここではＣＡＬＬ命令で、００Ａ２番地からのルーチンにジャンプさせています。００Ａ２番地のメモリからは、テキスト画面に１文字



表示するマシン語のプログラムが内蔵されています。したがって、CALL　00A2Hで、00A2番地のメモリに飛んで、1文字テキスト画面に表示するマシン語のプログラムを実行すると、1行目でAレジスタに入れて、F3DC番地のメモリに記憶させた0DH（●を表示する行座標）と、3行目でAレジスタに入れて、F3DD番地のメモリに記憶させた（●を表示する列座標）の位置に、5行目でAレジスタに入れた85H（●のキャラクタコード）で、●を画面の中央に表示することになります。

さて、最後のC9です。C9をアセンブリ言語で表わすと、

C9 ➡ RET

となります。C9のアセンブリ言語がRETです。RETは命令で、英語のRETURN（リターン）の略記号です。英語のRETURNは、「戻れ」とか「返れ」という意味です。マシン語のプログラムを実行したあと、このRET命令で、マシン語モニタに戻ります。

MSXパソコンでは、ベーシックでできたマシン語のモニタで、マシン語のプログラムをサブルーチンとして処理しますから、マシン語のプログラムの終わりには、かならずRET命令をおきます。

なおここではアセンブリ言語の説明の都合上、マシン語をアセンブリ言語になおしましたが、マシン語のプログラムは、アセンブリ言語でプログラムを書いてから、マシン語になおします(136頁参照)。

ニーモニック

ニーモニックとは

17頁でアセンブリ言語について説明しました。

アセンブリ言語は、16進数で書き表わされたマシン語の命令では私たちにわからないので、私たちにわかるように、英語の略記号で書き表わすようにしたものでした。

たとえば、ただ単に、テキスト画面に♥（ハート）を表示するプログラムをマシン語で書き表わすと、つぎのようになります。

```
3E 81 ←────── Aレジスタに81Hを入れよ
CD A2 00←00A2番地からの内部ルーチンにジャンプせよ
C9 ←────────── 戻れ
```

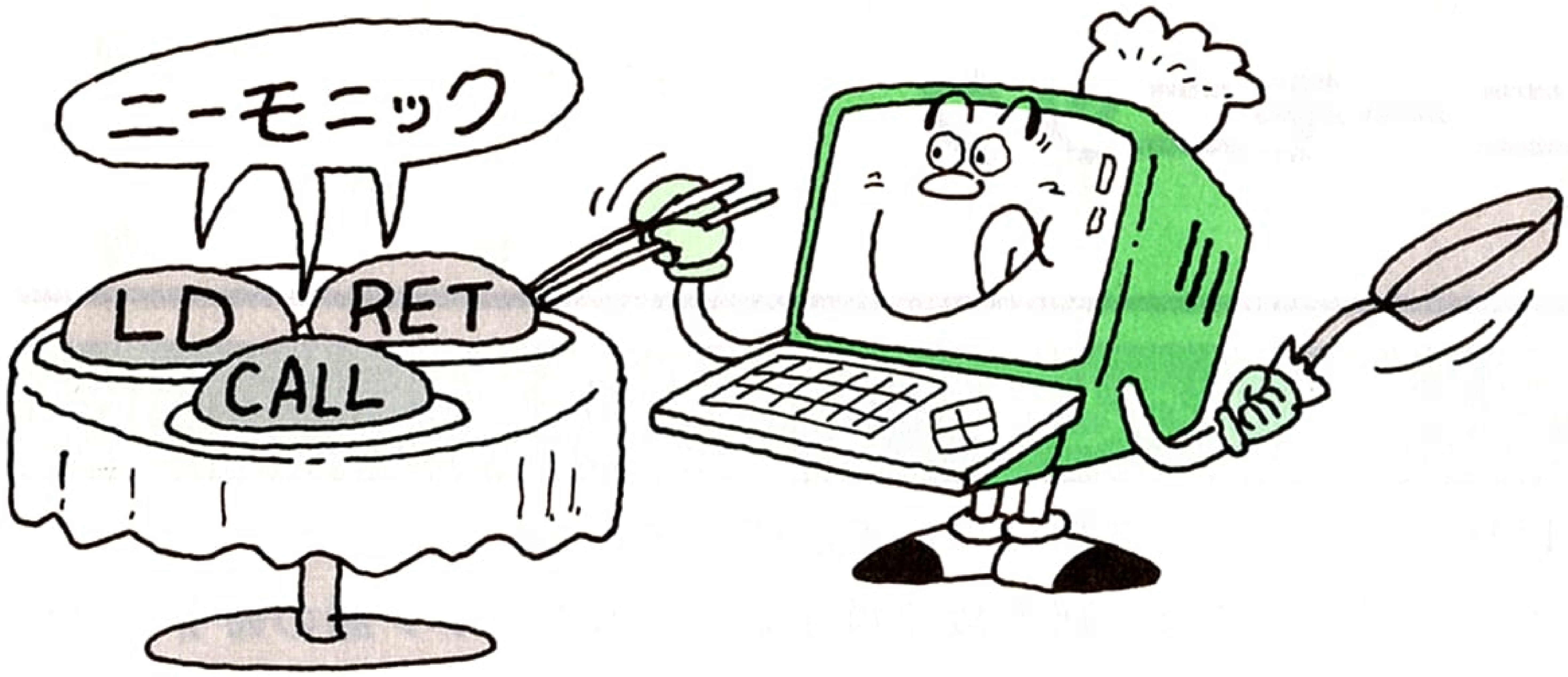
これでは私たちに命令の働きがわからないので、アセンブリ言語で書き表わすわけです。

```
LD A, 81H←──── Aレジスタに81Hを入れよ
CALL 00A2H←00A2番地からのルーチンにジャンプせよ
RET ←──────────── 戻れ
```

ＬＤは、英語のLoad（ロード）の略記号で、**ロード命令**といいます。CALLは、英語のCALL（コール）そのものですが、**サブルーチン・コール命令**または**コール命令**ともいいます。ＲＥＴは、英語のRETURN（リターン）の略記号で、**リターン命令**といいます。

さてニーモニックですが、ニーモニックは、この命令の働きを示した英語の略記号のことをいいます。つまり、

```
LD   ┐
CALL ├ ニーモニックはマシン語の命令を
RET  ┘ 英語の略記号で表わしたもの
```



などを、ニーモニックというわけです。

このように、命令の働きを示す英語の略記号を見ると、アセンブリ言語も、ニーモニックもおなじです。おなじである理由は、ニーモニックに基づいて、アセンブリ言語が作られているからです。

もちろん今日では、ＬＤという命令の働きを示す英語の略記号ばかりでなく、

```
LD  A, 81H
```

←命令の全体をニーモニックともいう

という命令の全体を、ニーモニックということもあります。またニーモニックとアセンブリ言語を、おなじ意味の言葉として使っている場合もあります。したがって、

```
LD   A, 81H
CALL 00A2H
RET
```

プログラム全体をもニーモニック・プログラムという

というアセンブリ言語のプログラムを、ニーモニック・プログラムということもあります。

このように、アセンブリ言語はニーモニックに基づいてできていますから、アセンブリ言語、ニーモニックのどちらをおぼえても、マシン語をおぼえるということに代わりありませんが、この本では、アセンブリ言語でマシン語をおぼえることにしています。

マシン語モニタ

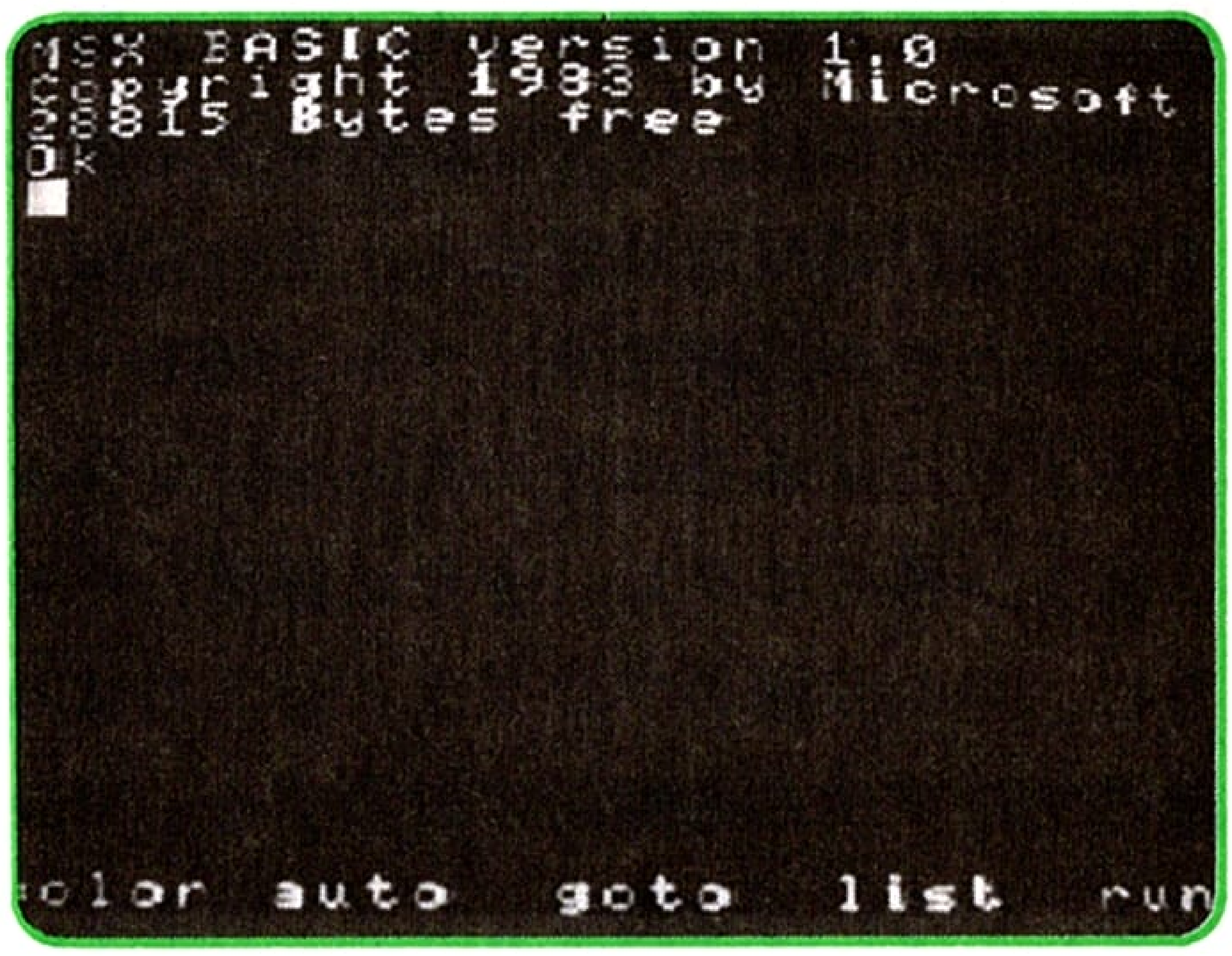
モニタとは

モニタとは、モニタ・プログラムのことをいいます。モニタ・プログラムを略して、モニタといっています。

では、モニタ・プログラムとはなんでしょうか。

ＭＳＸパソコンは機械です。したがって、ＭＳＸパソコン自体ではなにもすることができません。ＭＳＸパソコンに電源を入れても、ただ電気が流れるだけで、なにもすることができないのです。

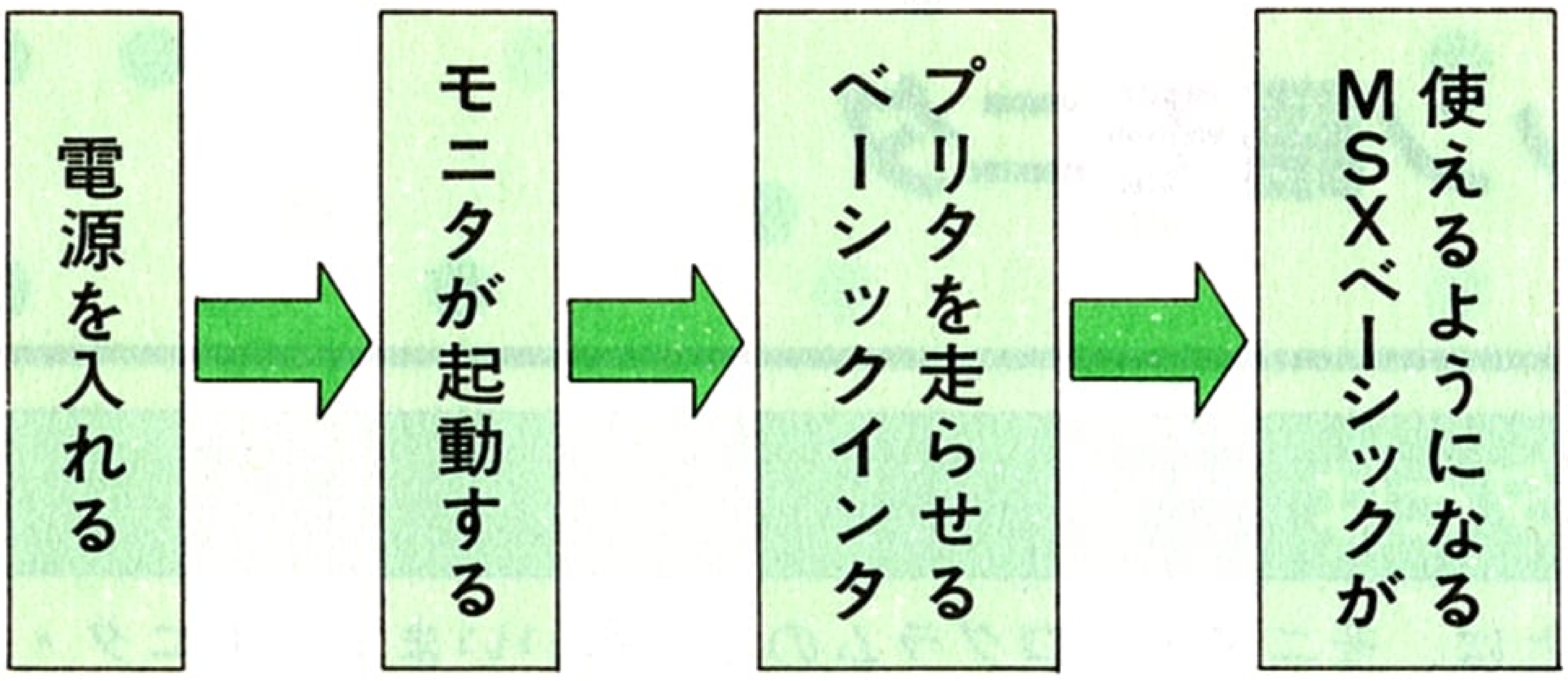
私たちがＭＳＸパソコンに電源を入れると、つぎの写真に示すように、画面に英語で文章が表示されるとともにＯＫが表示されて、ＭＳＸベーシックの命令を入力することができるようになります。これは、ＭＳＸパソコンに電源を入れたら、「ＭＳＸベーシックを使えるようにする」というプログラムが、ＭＳＸパソコンのなかに内蔵されているからです。このような働きをするプログラムを、**モニタ・プログラム**といいます。



モニタ・プログラムには、キーボードからのＭＳＸベーシックの入力、メモリに記憶されているプログラムなどの画面への表示、カセットテープへのセーブ・ロードなど、いろいろな働きがあります。

私たちが入力したプログラムによる動作以外のＭＳＸパソコンの動作は、すべてモニタ・プログラムによるものなのです。

つぎに、MSXベーシックが使えるようになる手順を図に示します。



マシン語モニタとは

さて、MSXパソコンはスイッチ・オン・ベーシックですから、電源を入れると同時にモニタが働いて、上の図に示したようにベーシックインタプリタを走らせて、MSXベーシックが使える状態にします。MSXベーシックが使える状態を、**ベーシックモード**といいます。

ベーシックモードでは、マシン語の命令を使ってMSXパソコンを操作することができません。マシン語の命令を使ってMSXパソコンを操作するには、マシン語の命令が使える状態、つまり**マシン語モード**にしなければなりません。それには、ベーシックの場合とおなじようにモニタが必要です。マシン語モードにして、マシン語の命令が使えるようにするモニタのことを、**マシン語モニタ**と呼んでいます。

マシン語モニタも、さきに説明したモニタ・プログラムとおなじように、マシン語のプログラムを入力する、メモリに記憶させたマシン語のプログラムを画面に表示する、マシン語のプログラムを実行するなど、いろいろな働きをもっています。

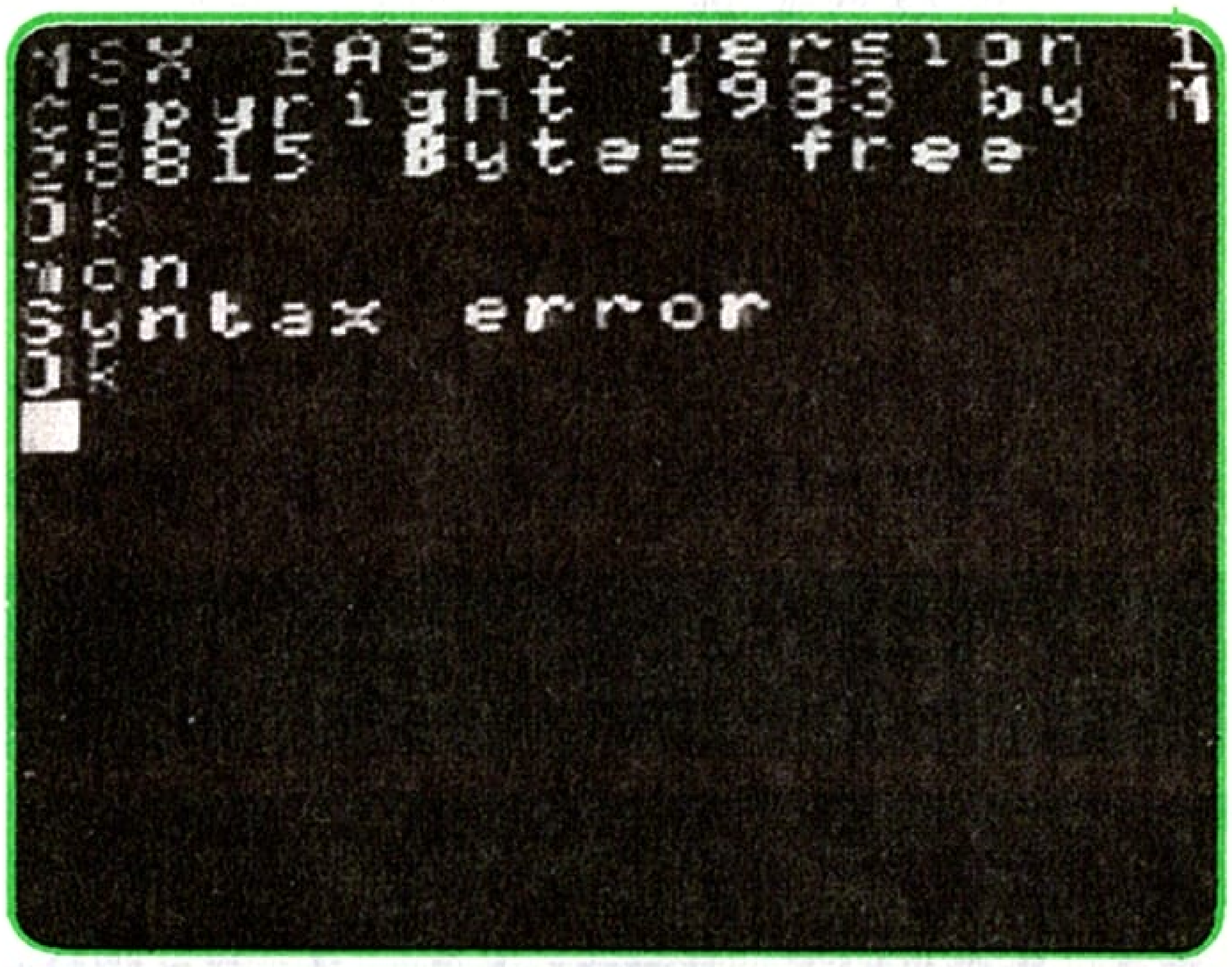
ところでMSXパソコンは、電源を入れるとベーシックモードになるようになっていますから、ベーシックモードからマシン語が使えるマシン語モードにする必要があります。これを、**「マシン語モニタを呼びだす」**といいます。

ベーシックモードからマシン語のモニタを呼びだすには、ふつう**MONコマンド**を使います。

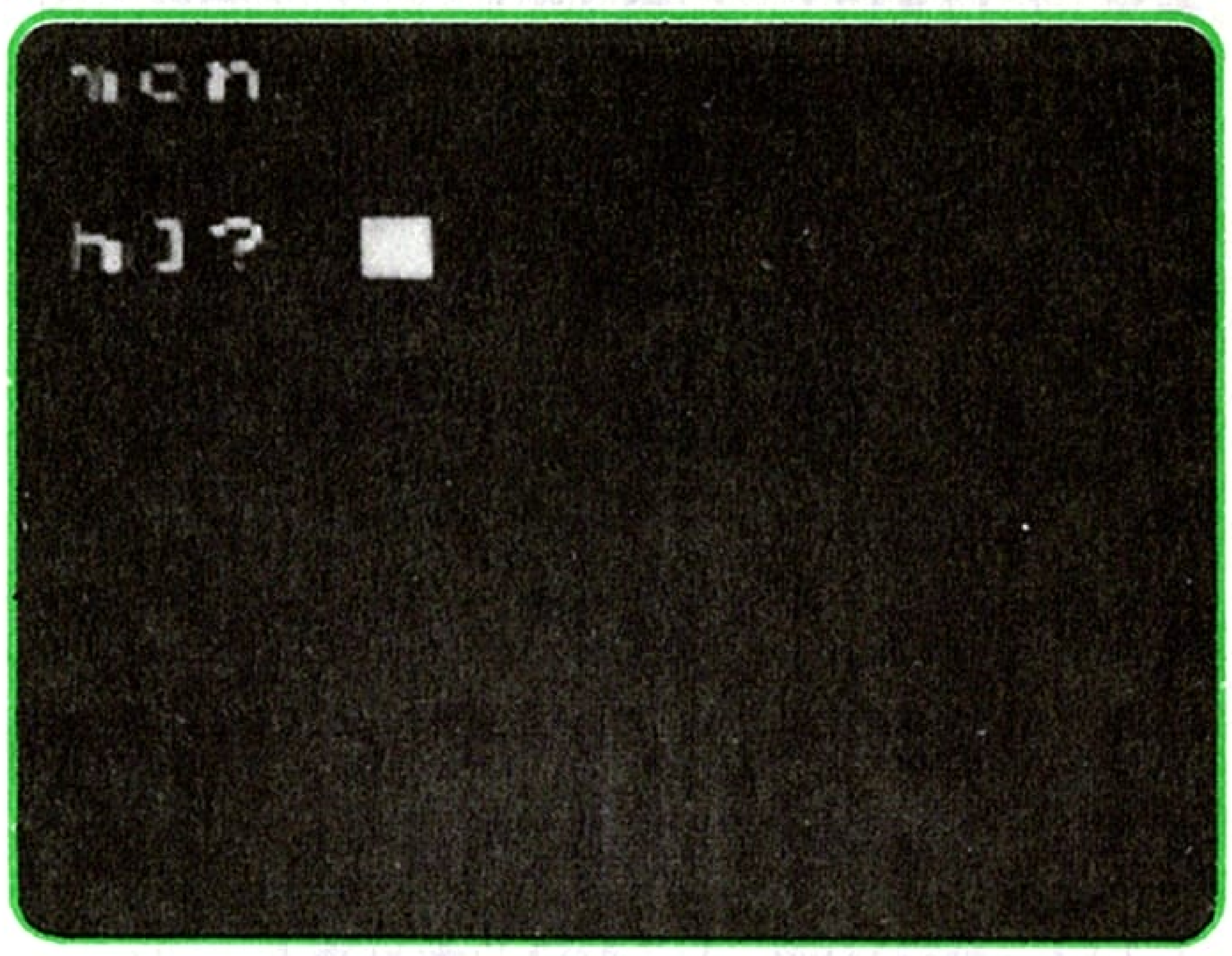
MONと入力して、⏎（リターンキー）を叩けばよいのです。

MON ⏎でSyntax　errorを表示する

では、MONと入力して⏎（リターンキー）を叩いてみます。すると、つぎの写真に示すように、MSXパソコンでは、Syntax errorを表示してしまいます。



MSXパソコン以外の機種では、どうなるでしょうか。ここでは、PC−8801 mkIIで行ってみます。MONと入力して⏎キーを叩くと、右の写真に示すように、入力したMONの下にh］を表示します。表示されたh］を**プロンプト**といいます。



このh］は、MONコマンドでマシン語モニタが呼びだされて、マシン語モードになったという印です。つまり、マシン語の命令が使える状態になったということです。

ところで、どうしてMSXパソコンでは、MONコマンドを入力して⏎キーを叩くと、Syntax errorになるのでしょうか。それは、ふつうMSXパソコンには、マシン語モニタが内蔵されていないからです。マシン語モニタが内蔵されているMSXパソコンであれば、MON ⏎でマシン語モードになるはずです。

さて、マシン語モニタが内蔵されていないMSXパソコンは、マシン語モニタを作らなければなりません。マシン語モニタは、ベーシッ

クで作ることができます。どんなベーシックの命令を使えば、マシン語モニタを作ることができるのかというと、つぎの、

ＰＥＥＫ関数←――メモリの内容を読みだす
ＰＯＫＥ文←――メモリにデータを書き込む
ＵＳＲ関数←――メモリに記憶させたマシン語を実行する

ベーシックの命令を使えばよいのです。このベーシックの命令の働きについては、ベーシックのマニュアルなどを参照してください。ここでは説明を省略します。つぎに簡単なマシン語モニタを示します。

簡単なマシン語モニタ

```
10 KEY OFF
20 PRINT "ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ";
30 INPUT A$
40 IF A$="d" THEN 80
50 IF A$="s" THEN 210
60 IF A$="g" THEN 430
70 END
80 REM "ﾀﾞﾝﾌﾟ ﾘｽﾄ"
90 PRINT "ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ";
100 INPUT B$,C$
110 B=VAL("&h"+B$)
120 C=VAL("&h"+C$)
130 PRINT RIGHT$("0000"+HEX$(B),4);"  ";
140 FOR I=B TO B+7
150 DT=PEEK(I)
160 PRINT RIGHT$("00"+HEX$(DT),2);" ";
170 NEXT
180 PRINT
```

```
190 IF I<=C THEN B=B+8:GOTO 130
200 GOTO 20
210 REM "ｾｯﾄ ﾒﾓﾘ"
220 PRINT "ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ";
230 INPUT B$
240 B=VAL("&h"+B$)
250 PRINT HEX$(B);"  ";
260 I=B:U=0
270 DT=PEEK(I)
280 PRINT RIGHT$("00"+HEX$(DT),2);"-";
290 H$=INKEY$:IF H$="" THEN 290
300 IF H$=" " THEN PRINT "  ";:GOTO 380
310 IF H$=CHR$(13) THEN PRINT:GOTO 20
320 PRINT H$;
330 J$=INKEY$:IF J$="" THEN 330
340 PRINT J$;
350 D$=H$+J$
360 DA=VAL("&h"+D$)
370 POKE I,DA
380 PRINT " ";
390 U=U+1:IF U<>4 THEN I=I+1:GOTO 270
400 PRINT
410 B=B+4
420 GOTO 250
430 REM "ｺﾞｰ"
440 PRINT "ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ";
450 INPUT B$
460 CLS
470 B=VAL("&h"+B$)
```

```
480 DEF USR=B
490 A=USR(0)
500 PRINT
510 PRINT
520 PRINT
530 GOTO 20
```

ここで示したマシン語モニタは、簡単なもので、

マシン語をメモリに入力する
メモリに入力したマシン語を取りだして画面に表示する
メモリに入力したマシン語を実行する

という、3つの機能をもつものです。この3つの機能をもっていれば、この本で使用するマシン語モニタとして十分だからです。

マシン語モニタは、ＭＳＸパソコンの雑誌などに掲載されていますが、そのマシン語モニタをもっていれば、そのマシン語モニタを使ってもかまいません。

このマシン語モニタを使って、マシン語を入力したり、メモリに入力したマシン語を画面に表示したり、実行させたりする方法については、33頁の「マシン語のプログラムを入力する」、40頁の「マシン語のプログラムを実行する」、45頁の「メモリの内容を画面に表示させる」、52頁の「入力したマシン語のプログラムを訂正する」、で説明することにします。



マシン語のプログラムを入力する

マシン語を入力できるメモリは?

30頁に簡単なマシン語モニタを示しました。ここでは、そのマシン語モニタを使って、マシン語のプログラムを入力することにします。つぎに示すのが、ここで入力するマシン語のプログラムです。このマシン語のプログラムについては、19頁で説明しています。

```
3E 0D
32 DC F3
3E 10
32 DD F3
3E 85
CD A2 00
C9
```

さて、マシン語のプログラムの入力とは、つぎのようにマシン語のひとつひとつを、ひとつひとつのメモリに記憶させることです。

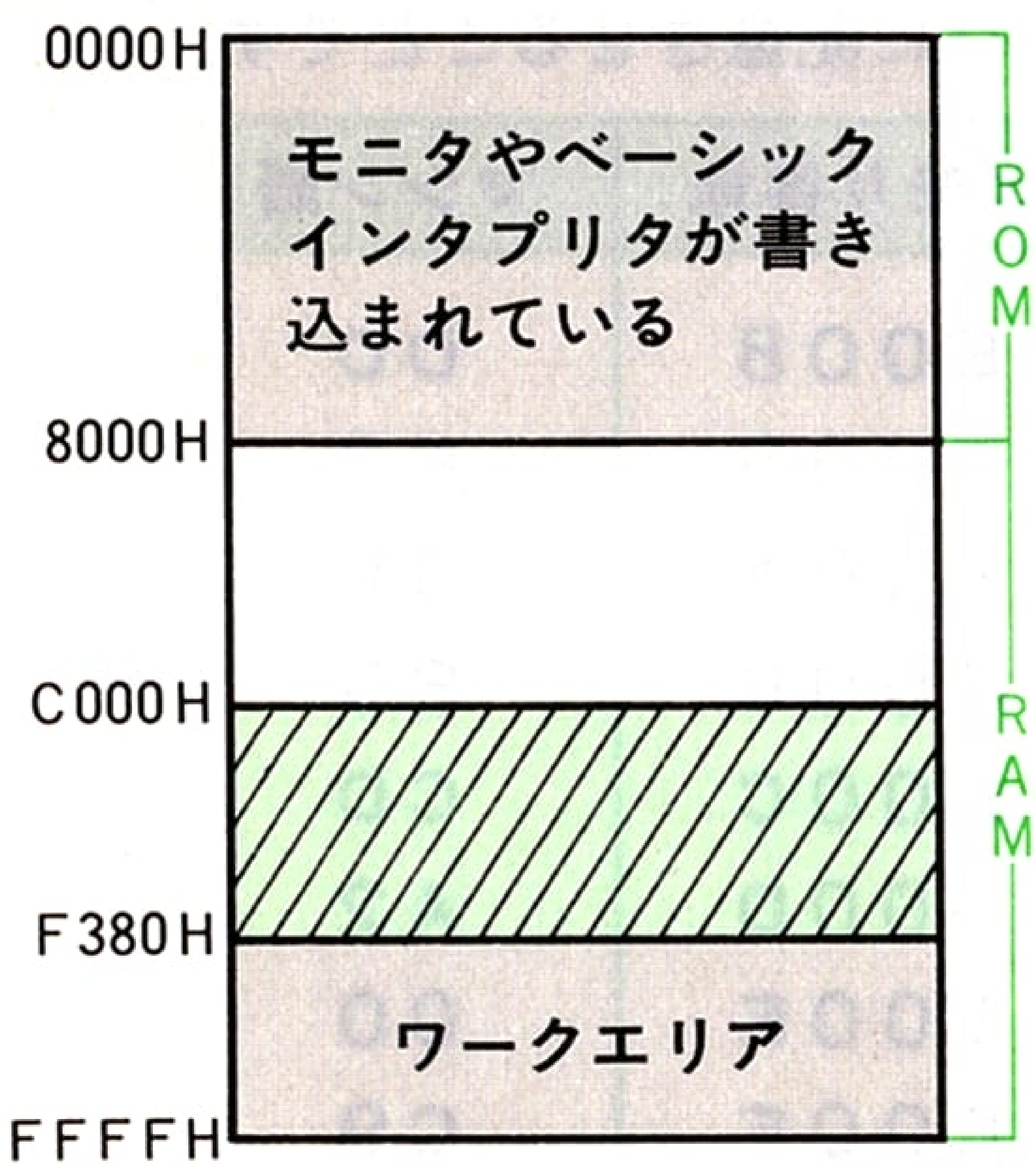
メモリ番地	マシン語	メモリ番地	マシン語
E000	3E	E008	DD
E001	0D	E009	F3
E002	32	E00A	3E
E003	DC	E00B	85
E004	F3	E00C	CD
E005	3E	E00D	A2
E006	10	E00E	00
E007	32	E00F	C9

この場合はマシン語のプログラムを、E000番地のメモリから入力していきますが、どの番地のメモリからでも、マシン語のプログラムを入力し自由に使うことができるのかというと、そうではありません。なぜなら私たちが、マシン語を入力して使うことができるメモリの範囲が決まっているからです。

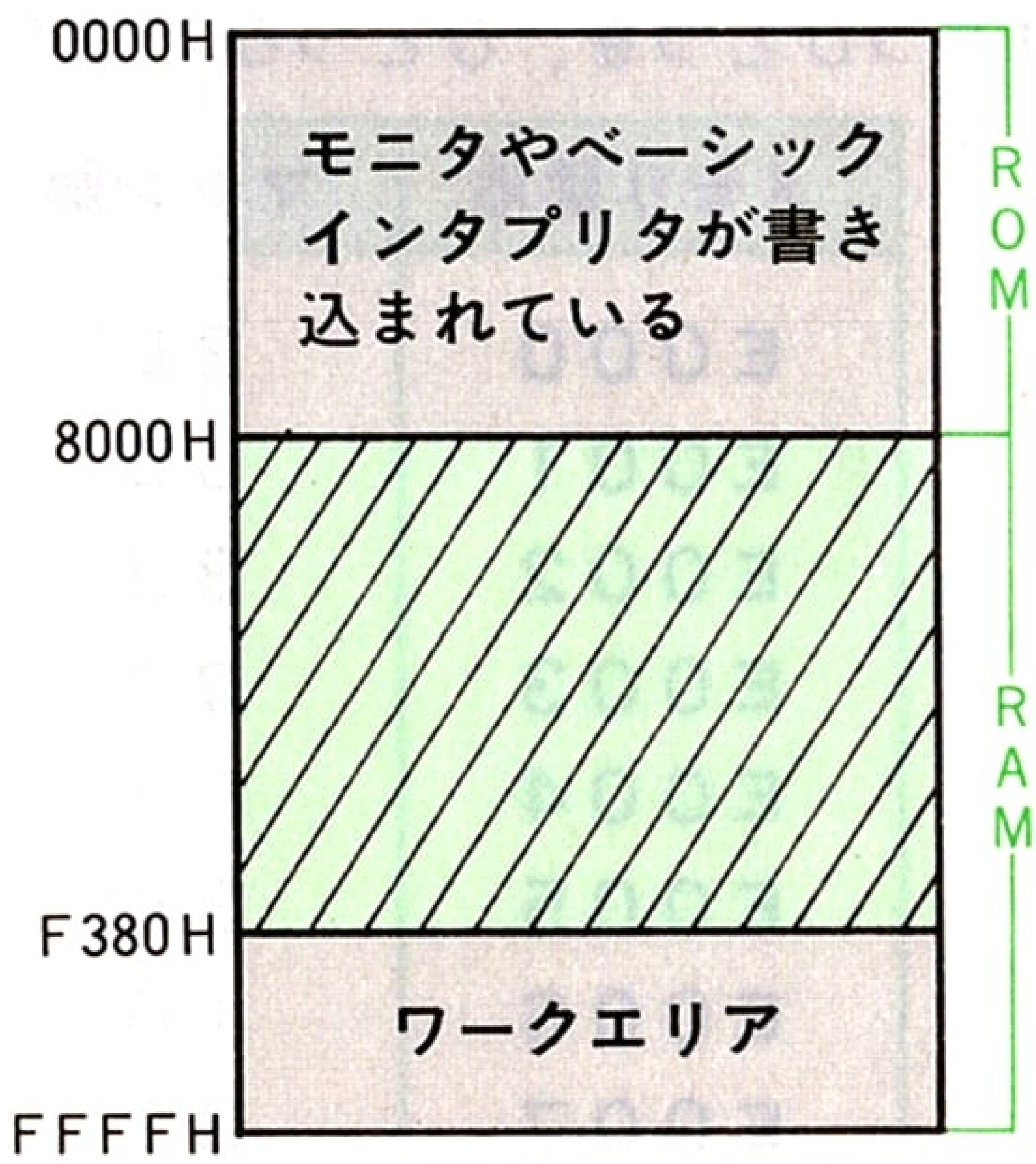
どの範囲のメモリを使うことができるかは、メモリ・マップを見るとわかります。メモリ・マップとは、メモリがどう使われているかを示した図のことです。参考までに、RAM（ラム）16Kバイトのメモリ・マップと、RAM32Kバイトのメモリ・マップを下に示します。左側の図が、RAM16Kバイトのメモリ・マップです。右側の図が、RAM32Kバイトのメモリ・マップです。

0000番地から8000番地のひとつまえの7FFF番地までは、ROM（ロム）領域ですから、ここは使うことができません。ROM領域には、モニタやベーシックインタプリタなどが記憶されているばかりでなくて、ROMというメモリICが使われているからです。このROM（Read Only Memory〈リード オンリー メモリ〉）というメモリICは、読みだし専用のメモリICで、書き込むことができません。

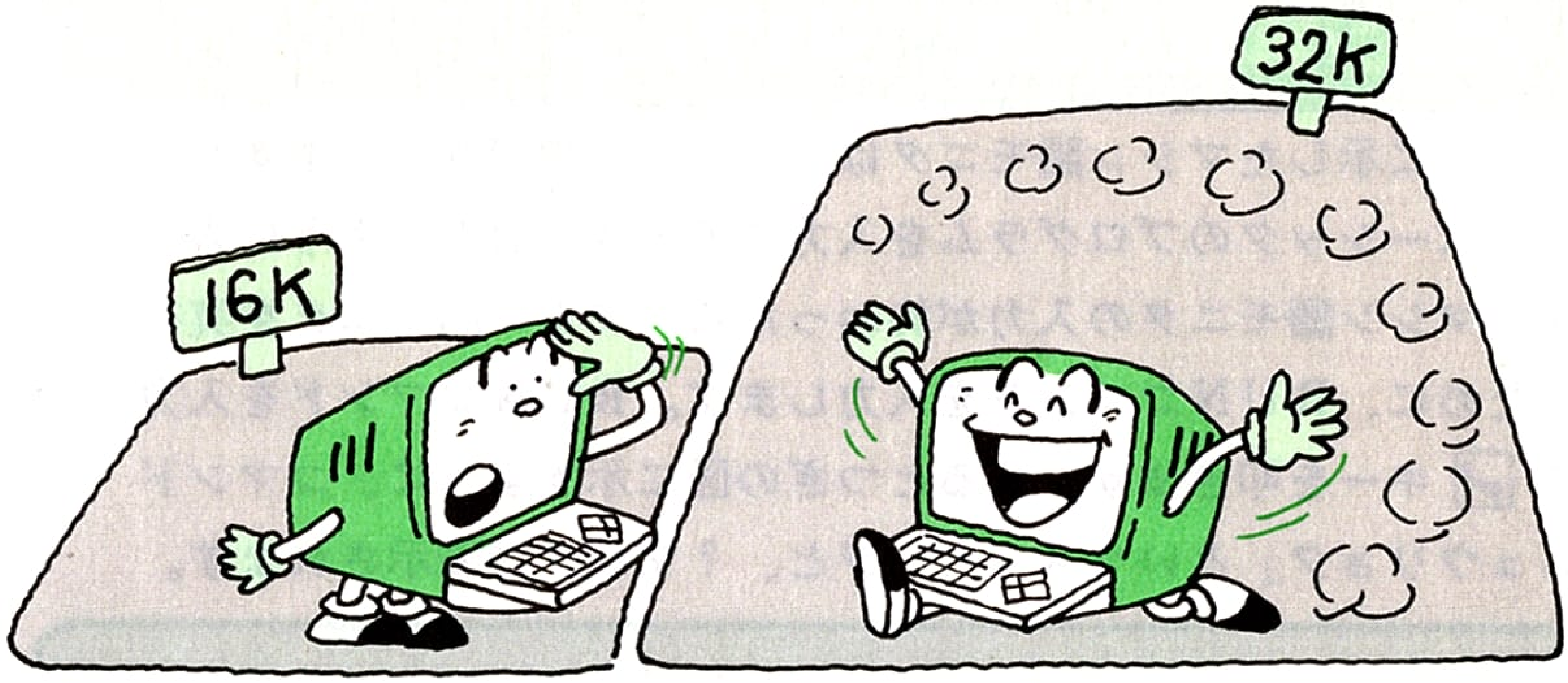
私たちがマシン語のプログラムを入力して使うことができるメモリは8000番地からFFFF番地までのRAM領域となります。RA



RAM16Kバイト



RAM32Kバイト



M（Random Access Memory）は、読みだし書き込みが自由にできるメモリＩＣです。つまり、私たちがマシン語のプログラムを入力したり、入力したマシン語のプログラムを、取りだしたりすることができるメモリということです。

ただ、８０００番地からＦＦＦＦ番地までを、すべて使うことはできません。なぜかというと、Ｆ３８０番地からＦＦＦＦ番地までは、ＭＳＸパソコン自体が仕事をするために使っているからで、そのＲＡＭ領域を、ワークエリアと呼んでいます。

したがって、私たちがマシン語のプログラムを入力して使うことができるメモリは、ＲＡＭ領域のワークエリアを除いた部分、つまりＲＡＭ16Ｋバイトの場合は、斜線が引いてあるＣ０００番地からＦ３８０番地のひとつまえのＦ３７Ｆ番地まで、ＲＡＭ32Ｋバイトの場合は、斜線が引いてある８０００番地からＦ３７Ｆ番地までとなります。

ＲＡＭ16Ｋバイト＝Ｃ０００番地からＦ３７Ｆ番地
ＲＡＭ32Ｋバイト＝８０００番地からＦ３７Ｆ番地

これで、私たちがマシン語のプログラムを入力して使うことができるメモリが、ＲＡＭ領域の何番地から何番地までであるかがわかりました。つぎは、マシン語モニタを入力して、マシン語のプログラムを入力することにします。

マシン語の入力は、Sコマンド

30頁に示したマシン語モニタはベーシックでできていますから、ふつうベーシックのプログラムを入力するときとおなじように入力します。マシン語モニタの入力が終わったら、マシン語モニタを実行させるために、RUNコマンドを入力します。RUNコマンドを入力したら ⏎ キーを叩きます。するとつぎの図に示すように、「コマンド　ノ　ニュウリョク」というメッセージと、?マークが表示されます。

```
run
コマンド゛ ノ ニュウリョク?
```

そうしたら、つぎの図に示すように、「S」と入力します。

```
run
コマンド゛ ノ ニュウリョク? s ←Sコマンドのsを入力する
```

「S」とは、**Sコマンド**のことです。

Sコマンドは、SET　MEMORY（セットメモリ）のことで、その頭文字のSをとってSコマンドといいます。

Sコマンド＝　SET　MEMORY（セットメモリ）

SETとは、「置く」とか「配置する」とかの意味です。MEMORYはメモリですから、「メモリに置く」、いいかえれば、メモリに入れるです。

ふつうSコマンドの働きを、メモリの内容を変更するとしています。

Sコマンド＝メモリの内容を変更する

なぜSコマンドの働きを、このようにいうのかはあとでわかります。ここでは、マシン語を入力するときは、Sコマンドを使うということをおぼえておいてください。

Sコマンドを入力したら、⏎ キーを叩きます。⏎ キーを叩くと、

つぎの図に示すように、「メモリバンチ　ノ　ニュウリョク」というメッセージと？マークが表示されます。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ?
```

「メモリバンチ　ノ　ニュウリョク」には、マシン語のプログラムを入力する最初のメモリ番地を入力します。

メモリバンチ　ノ　ニュウリョク？＝入力する最初のメモリ番地

この場合は、33頁でプログラムの最初のマシン語にＥ０００番地を割りあてていますから、つぎのようにＥ０００と入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
```

プログラムの最初のメモリ番地を入力する

Ｅ０００番地を入力したら、⏎キーを叩きます。

すると、つぎの図に示すようにＥ０００番地を表示して、そのつぎに００－を表示します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
E000 00-
```

E000番地のメモリの内容00－を表示する

Ｅ０００のつぎに表示された００－の００は、Ｅ０００番地のメモリに記憶されている内容です。私たちが、メモリのなかになにも入力していないときは、メモリの内容は、００かＦＦとなっていますから、その００かＦＦが表示されるのです。

```
E000  00 ⎫ なにもメモリに入力していないとき
E190  FF ⎭ メモリの内容は00かFF
```

メモリにマシン語を入力していたとすると、入力していたマシン語が表示されます。たとえば、E000番地のメモリに、0Dを入力していたとすると、E000番地のメモリの内容は0Dになっていますから、その0Dが表示されることになります。

```
E000  0D- ←――― メモリに記憶されている0Dを表示
```

さて、説明をまえに戻します。マシン語の入力は、つぎの図に示すように、表示された00－の－の右側に行います。この場合は、33頁に示したように、E000番地のメモリにはマシン語の3Eを入力しますから、3Eを00－の右側に入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
E000 00-3e 00-
```

3Eを入力すると、すぐつぎのメモリ番地の00－を表示

これで、E000番地のメモリに3Eを入力することができたわけです。

つまり、最初にE000番地のメモリに記憶されていた00に代えて、入力した3Eを記憶したことになります。

36頁でSコマンドの働きを、ふつうは「メモリの内容を変更する」こととしていると説明しましたが、それはいま説明したように、Sコマンドを使って、それまでメモリに記憶されていた00に代えて3Eを記憶させるからです。

さて、E000番地に3Eを入力すると、うえの図に示したように、自動的にそのつぎの、E001番地のメモリの内容00が表示されます。したがってつぎに入力する0Dを、つぎの図に示すように入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
E000 00-3e 00-0d 00-
```

00-に対してつぎつぎに
マシン語を入力していく

このようにして、表示されたメモリの内容００にたいしてマシン語を入力していきます。33頁に示したプログラムの最後のマシン語は、Ｃ９です。最後のマシン語Ｃ９を、Ｅ００Ｆ番地のメモリに入力すると、つぎの図のように、Ｅ００Ｆ番地のつぎの、Ｅ０１０番地のメモリの内容００を表示します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
E000 00-3e 00-0d 00-32 00-dc
E004 00-f3 00-3e 00-10 00-32
E008 00-dd 00-f3 00-3e 00-85
E00c 00-cd 00-a2 00-00 00-c9
E010 00-
```

つぎのE010番地の
内容00を表示する

ここで RET キー
を叩く

最後のマシン語C9を入力すると

最後のマシン語を入力し終わると、マシン語のプログラムの入力は終わったわけですから、つぎのメモリの内容００が表示されたところで、 ⏎ キーを叩きます。この場合は、うえの図に示すように、Ｅ０１０番地のメモリの内容００を表示したところで、 ⏎ キーを叩きます。 ⏎ キーを叩くとマシン語のプログラムの入力は終了となって、「コマンド　ノ　ニュウリョク」のメッセージと？マークが表示されます。これがマシン語の入力の仕方です。

マシン語のプログラムを実行させる

プログラムの実行は、Gコマンド

33頁で、マシン語のプログラムの入力の仕方について説明しましたから、ここでは、入力したマシン語のプログラムの実行です。ここで実行させるマシン語のプログラムは、33頁に示したマシン語のプログラムにします。

39頁でマシン語のプログラムの入力が終わったら、つぎのメモリ番地の００が表示されたところで、⏎キーを叩くと説明しました。⏎キーを叩くとつぎの図に示すように、「コマンド　ノ　ニュウリョク」というメッセージと？マークが表示されますから「G」と入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
E000 00-3e 00-0d 00-32 00-dc
E004 00-f3 00-3e 00-10 00-32
E008 00-dd 00-f3 00-3e 00-85
E00c 00-cd 00-a2 00-00 00-c9
E010 00-
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? g ←──GコマンドのGを入力
```

「G」とは、**Gコマンド**のことです。Gコマンドは、英語のＧＯ（ゴー）の頭文字で、マシン語のプログラムを実行させるコマンドです。

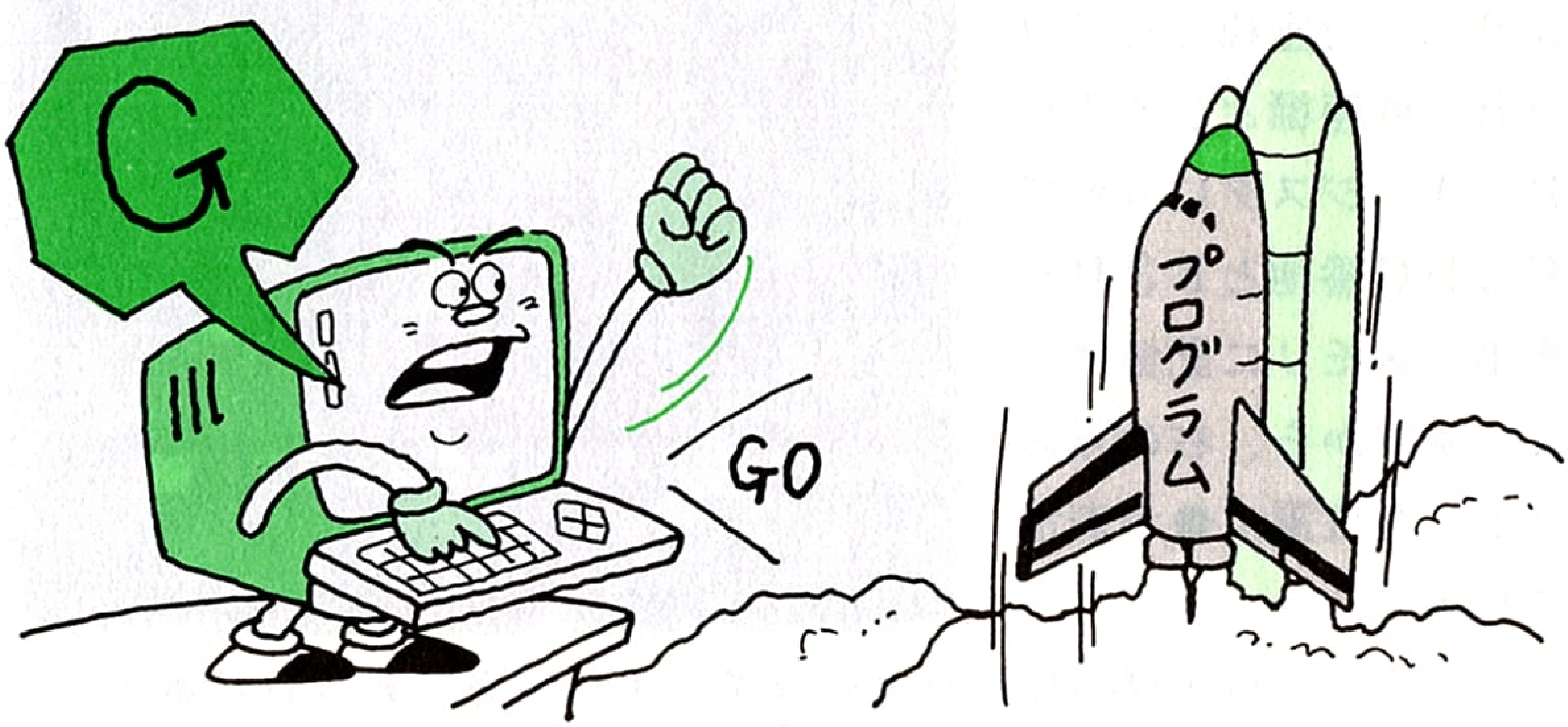
Gコマンド＝マシン語のプログラムを実行

Gコマンドを入力したら ⏎ キーを叩きます。⏎ キーを叩くと、つぎの図に示すように、「メモリバンチ　ノ　ニュウリョク」というメッセージと、？マークが表示されます。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
E000 00-3e 00-0d 00-32 00-dc
E004 00-f3 00-3e 00-10 00-32
E008 00-dd 00-f3 00-3e 00-85
E00c 00-cd 00-a2 00-00 00-c9
E010 00-
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? g
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ?
```

この場合の「メモリバンチ　ノ　ニュウリョク」には、マシン語のプログラムが記憶されている、最初のメモリ番地を入力します。つまり、実行させる最初のメモリ番地です。

メモリバンチ　ノ　ニュウリョク？＝実行させる最初のメモリ番地

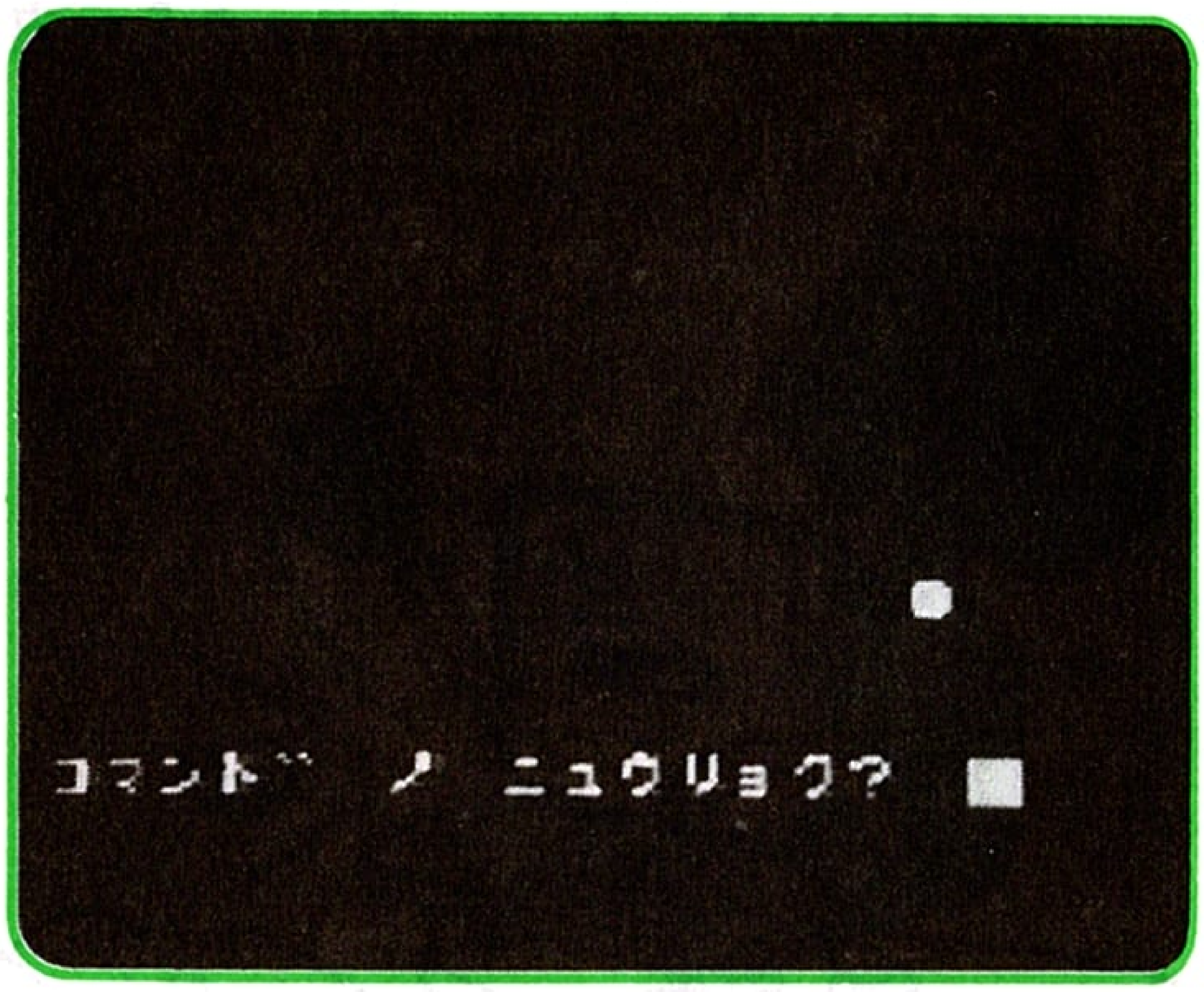


この場合はＥ０００番地のメモリから、マシン語のプログラムが記憶されています。したがって、実行させる最初のメモリ番地はＥ０００ですから、つぎの図に示すようにＥ０００と入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
E000 00-3e 00-0d 00-32 00-dc
E004 00-f3 00-3e 00-10 00-32
E008 00-dd 00-f3 00-3e 00-85
E00c 00-cd 00-a2 00-00 00-c9
E010 00-
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? g
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
```

実行させる最初の
メモリ番地を入力

Ｅ０００と入力して ⏎ キーを叩くと、Ｅ０００番地からのマシン語を取りだして実行していきます。つぎに、このプログラムの実行結果を写真で示します。



このマシン語のプログラムは、22頁で説明したように、行座標として０ＤＨ、列座標として１０ＨをＡレジスタに入れてＦ３ＤＣ番地とＦ３ＤＤ番地のメモリに記憶させていますから、右の写真に示した位置に●が表示されます。

ところで、行座標の０ＤＨと列座標の１０Ｈを、Ｆ３ＤＣ番地とＦ３ＤＤ番地以外のメモリに記憶させたらどうなるでしょうか。

行座標と列座標の値をほかのメモリに記憶させると

では、行座標の０ＤＨをＦ３１０番地のメモリに、列座標の１０Ｈ をＦ３２０番地のメモリに記憶させてみることにします。

行座標の０ＤＨをＦ３１０番地のメモリに記憶させるには、33頁の最初に示したマシン語のプログラムの2行目を、

2行目　32　DC　F3　➡　32　10　F3

に変更します。列座標の１０ＨをＦ３２０番地のメモリに記憶させるには、プログラムの4行目を、

4行目　32　DD　F3　➡　32　20　F3

に変更します。

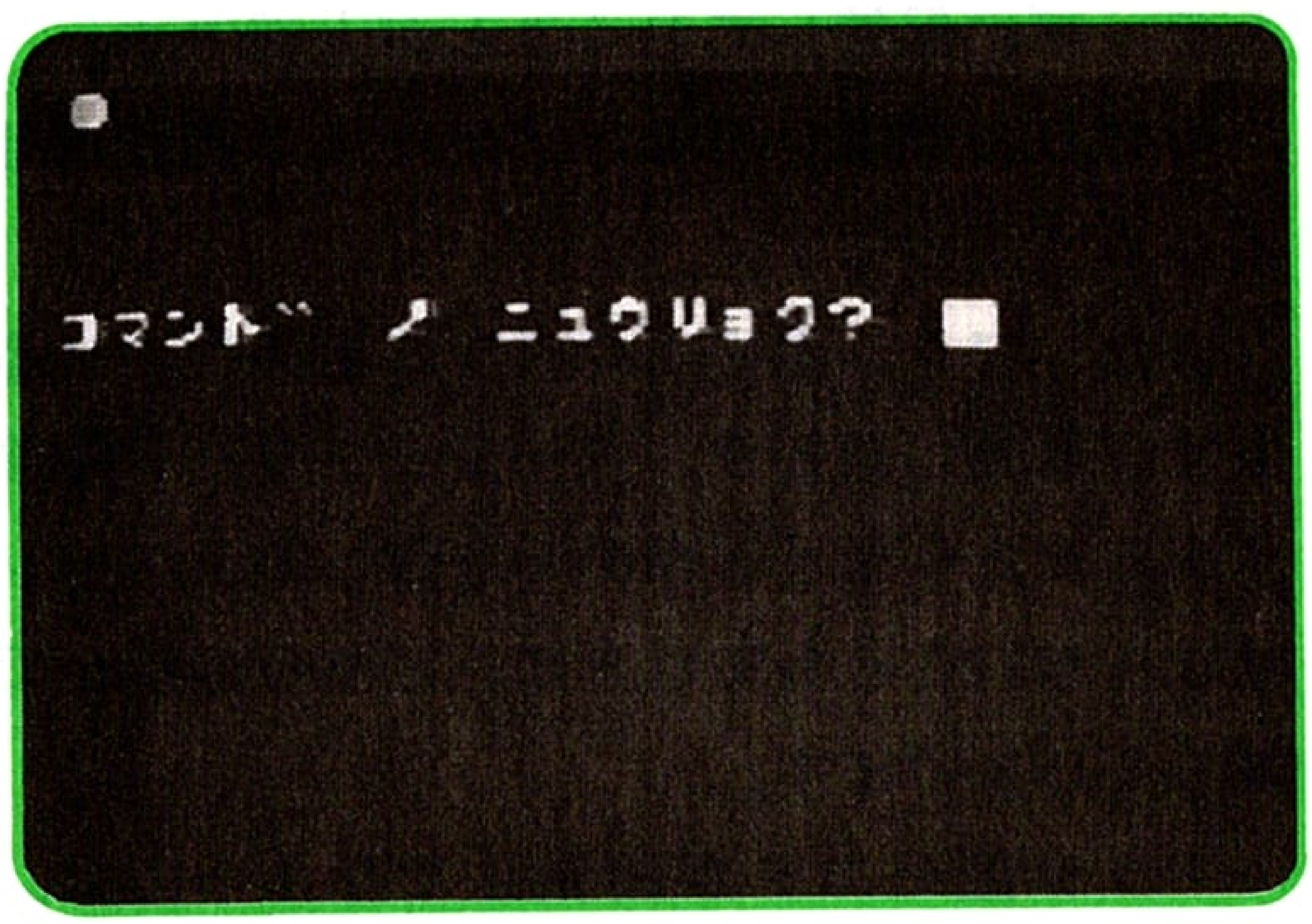
33頁のメモリ番地をつけたマシン語のプログラムの場合は、メモリ番地Ｅ００３のところを、

E003　DC　➡　E003　10

に、メモリ番地Ｅ００８のところを、

E008　DD　➡　E008　20

に変更します。このように変更して実行させると、つぎの写真に示すように、●が指定した行座標０ＤＨ、列座標１０Ｈの位置に表示されないで、テキスト画面の左上に表示されてしまいます。



これでは、行座標と列座標を指定した意味がありません。

どうして、このような結果になったのでしょうか。なぜなら、33頁の最初に示したマシン語のプログラムの6行目に、

CD　A2　00　➡　CALL　00A2H

とあります。マシン語のＣＤ　Ａ２　００は、アセンブリ言語では

ＣＡＬＬ　００Ａ２Ｈです。23頁でＣＡＬＬ　００Ａ２Ｈは、００Ａ２Ｈ番地にある、テキスト画面に１文字表示するルーチンにジャンプさせて、●をテキスト画面に表示させると説明しました。

このように００Ａ２番地にある、テキスト画面に１文字表示するルーチンを利用して●をテキスト画面に表示する場合、行座標を指定する値と列座標を指定する値は、

行座標を指定する値＝Ｆ３ＤＣ番地のメモリに記憶
列座標を指定する値＝Ｆ３ＤＤ番地のメモリに記憶

と決まっているからです。

したがって、行座標を指定する値と列座標を指定する値を、Ｆ３ＤＣ番地とＦ３ＤＤ番地以外のメモリ番地に記憶させると、その値は無効となってしまって、●がテキスト画面の左上に表示されることになるのです。

マシン語モニタは、かならずカセットテープへ

マシン語のプログラムを実行するまえにマシン語モニタは、カセットテープに保管しておくことが大切です。

なぜなら電源を切ると、メモリに記憶させたものは、すべて消滅してしまうからです。

また、マシン語のプログラムを実行させると、暴走することがあります。暴走したとき、ベーシックモードに戻ってしまうこともありますが、またリセットボタンを押して、ベーシックモードに戻さなければならないこともあります。どちらの場合でも、マシン語モニタは消滅してしまいます。

マシン語モニタが消滅してしまっては、なにもすることができませんから、ふたたびマシン語モニタを入力しなければなりません。簡単なマシン語モニタであっても、いちいちキーボードのキーを叩いて入力していたのでは大変です。そのため、カセットテープにマシン語モニタを記録しておく必要があるのです。

メモリの内容を画面に表示させる

画面に表示させるには、Dコマンド

33頁でマシン語のプログラムの入力の仕方について、40頁ではマシン語のプログラムの実行のさせ方について説明しました。ここでは、メモリの内容、つまりメモリに記憶されているものを画面に表示させることにします。

メモリに記憶されているものは、私たちが入力したマシン語のプログラムばかりではありません。13頁で説明したベーシックインタプリタや23頁で説明したルーチンなど、いろいろなものが記憶されています。それらを画面に表示させるわけです。

まず、30頁に示したマシン語モニタを入力して、RUNコマンドで実行させます。すると、つぎの図に示すように、「コマンド　ノ　ニュウリョク」というメッセージと、？マークが表示されます。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ?
```

そこで、つぎの図に示すように「D」と入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
```

DコマンドのDを入力する

「D」とは**Dコマンド**のことです。Dコマンドは、DUMP MEMORY（ダンプ・メモリ）の頭文字を取ったもので、メモリの内容を画面に表示する働きをします。

Dコマンド＝メモリの内容を画面に表示

Dコマンドを入力したら ⏎ キーを叩きます。すると、つぎの図に示すように、「メモリバンチ　ノ　ニュウリョク」のメッセージと？マークが表示されます。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ?
```

「メモリバンチ　ノ　ニュウリョク」には、画面に表示させるメモリの最初のメモリ番地と最後のメモリ番地を入力します。

メモリバンチ　ノ　ニュウリョク？＝〈画面に表示させる最初のメモリ番地〉，〈最後のメモリ番地〉

ROM領域の内容を表示させる

たとえば、００００番地から００２Ｆ番地のメモリの内容を表示させるには、００００と００２Ｆをコンマで区切って入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? 0000,002f
```

この場合、最初のメモリ番地だけ入力して ⏎ キーを叩くと、？マークをふたつ表示して、最後のメモリ番地の入力を求めてきます。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? 0000
??
```

??←最後のメモリ番地の入力を求めてくる

さて、０００００と００２Ｆを入力して⏎キーを叩くと、つぎの図に示すようにメモリの内容が画面に表示されます。

００００番地から００２Ｆ番地までのメモリはＲＯＭ領域ですから、ＲＯＭ領域に記憶されているベーシックインタプリタとか、そのほかのルーチンが表示されることになります。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? 0000,002f
0000 F3 C3 D7 02 BF 1B 98 98
0008 C3 83 26 00 c3 B6 01 00
0010 C3 86 26 00 C3 D1 01 00
0018 C3 45 1B 00 C3 17 02 00
0020 C3 6A 14 00 C3 5e 02 00
0028 C3 89 26 00 00 00 00 00
```

この場合は、ＲＯＭ領域の一部を表示させてみたわけですが、ＲＯＭ領域の最後までを画面に表示させてみるときは、ＲＯＭ領域は００００番地から、８０００番地のひとつまえの７ＦＦＦ番地までですから、つぎの図に示すように、００００と７ＦＦＦを入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? 0000,7fff
```

このようにして⏎キーを叩くと、ＲＯＭ領域に記憶されているインタプリタやルーチンが、画面につぎつぎに表示されていきます。

途中で表示を一時中止するときは、STOPキーを叩きます。

一時停止＝STOPキー

表示を再開するときは、もう一度STOPキーを叩きます。

表示を中止するときは、[STOP]キーと[CTRL]キーを一緒に押します。

表示の中止＝[STOP]キー＋[CTRL]キー

RAM領域の内容を表示させる

つぎはＲＡＭ領域のメモリの内容を、画面に表示させてみます。まず34頁のメモリ・マップで示した、ＭＳＸパソコン自体が仕事に使っているワークエリアからはじめます。ワークエリアはＦ３８０番地からＦＦＦＦ番地までですから、つぎのように入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? f380,ffff
```

このように入力して[⏎]キーを叩くと、つぎの図に示すように、ワークエリアの内容が表示されていきます。ここでは、スペースの関係から全部を示すことができないので、その一部を示します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? f380,ffff
F380 D3 A8 5E 18 03 D3 A8 73
F388 7A D3 A8 C9 D3 A8 08 CD
F390 98 F3 08 F1 D3 A8 08 C9
F398 DD E9 5A 47 5A 47 5A 47
F3A0 5A 47 5A 47 5A 47 5A 47
```

このようにワークエリアには、マシン語のプログラムが記憶されていて、ＭＳＸパソコン自体が仕事で使うようになっています。ここで注意することは、34頁でも説明したように、ワークエリアはＲＡＭ領

域にあるということです。

ＲＡＭはＲＯＭと違って書き込むことができるメモリです。つまり私たちが、マシン語のプログラムを入力することができるメモリですから、もし私たちが、ワークエリアにマシン語のプログラムを入力してしまうと、うえに示したワークエリアのマシン語のプログラムが書き換えられてしまいます。したがって、間違ってもワークエリアにマシン語のプログラムを入力してはならないのです。

マシン語のプログラムが記憶できるメモリ領域には？

つぎは、８０００番地から８３４７番地のメモリの内容を表示させてみることにします。８０００番地から８３４７番地は、ＲＡＭ32Ｋバイトの場合です。ＲＡＭ16Ｋバイトの場合は、８０００番地からＢＦＦＦ番地までのメモリがありませんから、Ｃ０００番地からＣ３４７番地となります。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? 8000,8347
```

RAM32Kの場合

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? c000,c347
```

RAM16Kの場合

このように入力して ⏎ キーを叩くと、ＲＡＭ32Ｋバイトの場合は、８０００番地から８３４７番地までのメモリの内容が表示されます。ＲＡＭ16Ｋの場合は、Ｃ０００番地からＣ３４７番地までのメモリの内容が表示されます。表示される内容はメモリ番地が異なるだけで、ＲＡＭ32Ｋバイトの場合もＲＡＭ16Ｋバイトの場合もおなじですから、ここでは、ＲＡＭ32Ｋバイトの場合の一部を示します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? 8000,8347
8000 00 09 80 0A 00 CC 20 FB
8008 00 1D 80 14 00 91 20 22
8010 BA CF FD C4 DE 20 69 6E
8018 70 75 74 22 00 26 80 1E
8020 00 85 20 41 24 00 39 80
```

このように、RAM32Kバイトの場合は、８０００番地から８３４７番地までのメモリの内容を表示させると、またRAM16Kバイトの場合は、Ｃ０００番地からＣ３４７番地までのメモリの内容を表示させると、16進数２桁の数字が表示されます。16進２桁というとマシン語とおなじですが、これはなにかというと、私たちが入力したMSXベーシックで作ったマシン語モニタです。

これでわかるように、MSXベーシックのプログラムは、RAM32Kバイトの場合は８０００番地のメモリから、またRAM16Kバイトの場合はＣ０００番地のメモリから、自動的に入力されて記憶されていきます。

ところでさきにみたように、MSXベーシックのプログラムも、メモリのなかでは、マシン語とおなじ16進数２桁で記憶されていますが、これはマシン語ではありません。

これはあくまでもベーシックです。この形をMSXベーシックの内部表現と呼んでいます。

さて、私たちが入力したマシン語モニタが記憶されているメモリに、マシン語のプログラムを入力するとこれも壊れてしまいますから、このようにDコマンドで、どの番地のメモリまでマシン語モニタが記憶されているかを調べて、それ以降の番地のメモリから、マシン語のプログラムを入力するようにしなければなりません。

入力したマシン語プログラムの表示

最後になりましたが、私たちが入力したマシン語のプログラムを、画面に表示させることにします。もう、メモリに記憶されている内容の表示の仕方はわかったことと思いますが、マシン語のプログラムに間違いがないかどうかを確認するとき、または訂正するときには、入力したマシン語のプログラムを、画面に表示させなければならないからです。

この場合は33頁で示したマシン語のプログラムが、Ｓコマンドを使ってＥ０００番地から、Ｅ００Ｆ番地までのメモリに記憶されていることにします。したがってつぎのように、マシン語のプログラムが記憶されている最初のメモリ番地Ｅ０００と、最後のメモリ番地Ｅ００Ｆを入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000,e00f
```

このように入力して ⏎ キーを叩くと、つぎに示すように、入力したマシン語のプログラムが画面に表示されます。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000,e00f
E000 3E 0D 32 DC F3 3E 10 32
E008 DD F3 3E 85 CD A2 00 C9
```

マシン語のプログラムの訂正の仕方については、52頁を参照してください。

これまで説明してきたように、Ｄコマンドはメモリの内容を画面に表示するコマンドで、いろいろなことに使います。

入力したマシン語のプログラムを訂正する

訂正するには、Sコマンド

ＭＳＸベーシックでは、ＭＳＸパソコンにプログラムを入力したあと、簡単にプログラムを訂正したり追加したり、また削除することができます。たとえばプログラムに間違いがあったら、カーソルコントロールキーでカーソルをその位置まで移動させて訂正することができます。ステートメントを追加するときは、追加するステートメントに、追加するところの間の行番号をつけて ⏎ キーを叩いておけば追加することができます。また削除するときは、削除するステートメントの行番号を入力して、 ⏎ キーを叩いておけば削除することができます。

マシン語では、ＭＳＸベーシックのようなわけにはいきません。マシン語でできることといえば、マシン語ひとつの訂正ぐらいで、あとはプログラムの入力のやりなおしとなります。ただそれにも方法があるので、ここでは、その方法について説明しておくことにします。

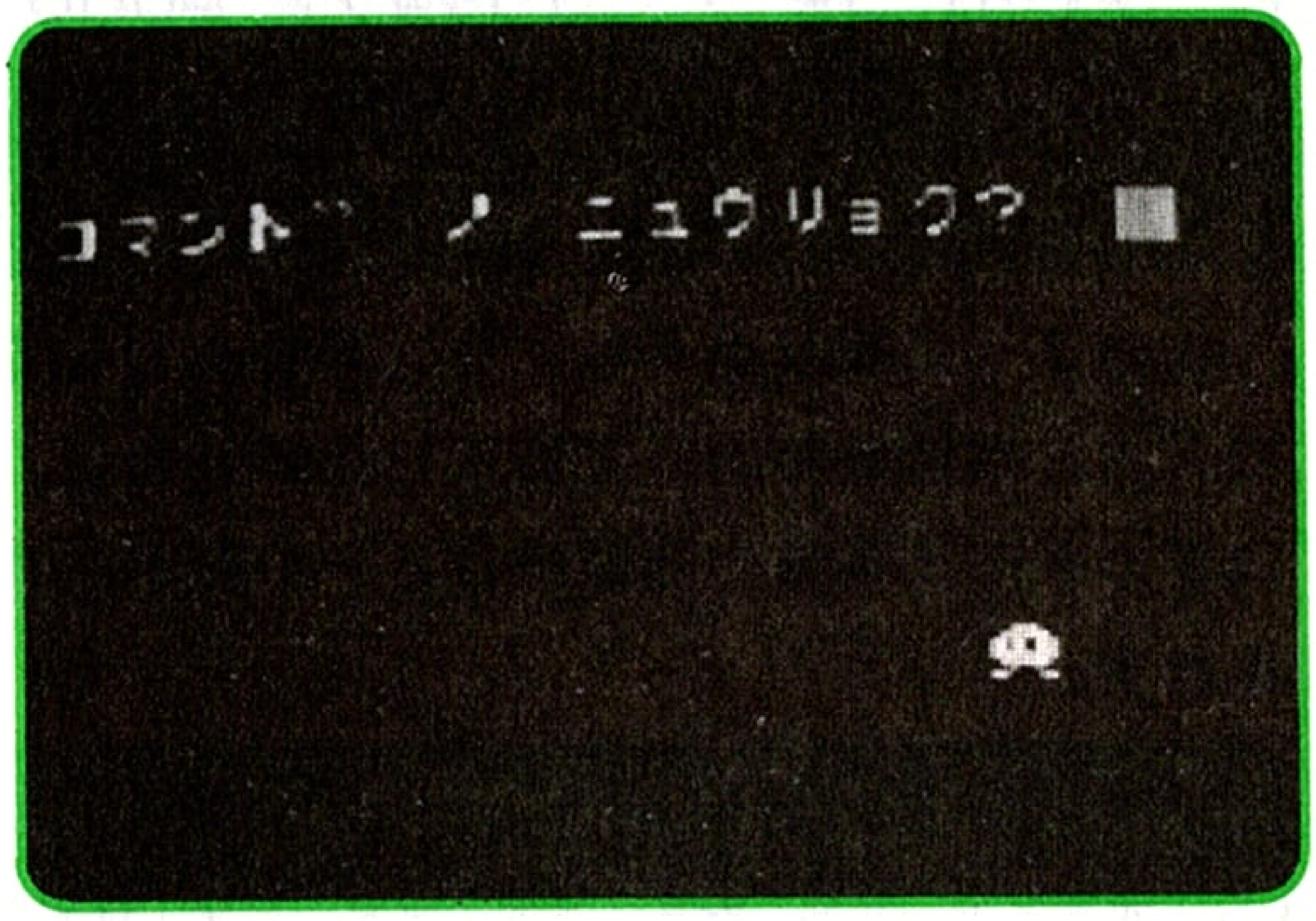


入力しているとき、入力ミスに気がついたときは

まず、ここで使うプログラムを示します。

メモリ番地	マ シ ン 語	アセンブリ言語
E000	21 50 E0	LD HL, E050H
E003	11 00 38	LD DE, 3800H
E006	01 08 00	LD BC, 0008H
E009	CD 5C 00	CALL 005CH
E00C	21 58 E0	LD HL, E058H
E00F	11 00 1B	LD DE, 1B00H
E012	01 04 00	LD BC, 0004H
E015	CD 5C 00	CALL 005CH
E018	C9	RET

E050	3C 7E D7	キャラクタのデータ
E053	D7 FF 7E	
E056	24 C3 60	
E059	80 00 0F	



このマシン語のプログラムは、19頁で説明したテキスト画面に●というキャラクタを表示するマシン語のプログラムと違って、右の写真に示すように、データで作ったキャラクタ（スプライト）を、スプライト画面に表示するプログラムです。

このプログラムは、164頁で説明します。

さて、30頁に示したマシン語モニタを入力したら、ＲＵＮコマンドでマシン語モニタを実行させます。そしてＳコマンドと、まえの頁に示したマシン語のプログラムにつけられている、最初のメモリ番地Ｅ０００を入力して ⏎ キーを叩きます。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
E000 00-
```

Ｓコマンドを入力

マシン語を入力する最初のメモリ番地を入力

⏎ キーを叩いて、Ｅ０００番地のメモリの内容００と－が表示されたら、マシン語の２１を入力します。２１を入力すると、つぎのＥ００１番地のメモリの内容００と－が表示されますから、つぎのマシン語の５０を入力します。５０を入力すると、つぎのＥ００２番地のメモリの内容００と－を表示しますから、そのつぎのマシン語Ｅ０を入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000
E000 00-21 00-50 00-e0 00-11
```

つぎつぎにマシン語を入力

このように繰り返えしマシン語を入力していって、途中でマシン語の入力ミスに気がついて訂正したいときは、つぎの図に示すように、つぎのメモリ番地の内容００と－が表示されたところで ⏎ キーを叩きます。

```
E000 00-21 00-50 00-e0 00-11
E004 00-00 00-37 00-
```

ここでRETキーを叩く

38を37と間違えて入力

この場合は、３８と入力するところを３７と間違って入力してしまいました。⏎キーを叩くと「コマンド　ノ　ニュウリョク？」となりますから、Ｓコマンドを入力して⏎キーを叩きます。

```
E000 00-21 00-50 00-e0 00-11
E004 00-00 00-37 00-
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ?
```

←RETキーを叩く
←Sコマンドを入力

すると、「メモリバンチ　ノ　ニュウリョク？」となります。この場合、２行目の最初の００－は、左側に示されているＥ００４番地のメモリの内容ですから、３７を入力した００－は、そのつぎのＥ００５番地の内容ということになります。

```
E004  00-00  00-47  00-
      ↑      ↑      ↑
```

Ｅ００４番地　Ｅ００５番地　Ｅ００６番地

したがって「メモリバンチ　ノ　ニュウリョク？」には、Ｅ００５と入力します。

```
E000 00-21 00-50 00-e0 00-11
E004 00-00 00-37 00-
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e005
```

⏎キーを叩くと、メモリ番地Ｅ００５が表示されて、間違って入力した３７と－が表示されます。３７－が表示されたら、３７－に対して正しいマシン語３８を入力すればよいのです。

```
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e005
E005 37-38 00-
```

E005と入力してRETキーを叩く
E005番地のメモリの内容37を表示する
37－に対して正しいマシン語38を入力する
38を入力すると、つぎのメモリ番地の内容を表示

何番地のメモリかわからないときは

ところで16進数に慣れていないと、訂正するところがわかっても、そのメモリが何番地なのかわからない、ということもあるかもしれません。つぎの場合は、53頁に示したマシン語のプログラムを全部入力したあと、入力ミスがないかどうかを確かめるために、Dコマンドで画面に表示したものです。

```
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000,e018
E000 21 50 E0 11 00 38 01 08
E008 00 CD 5C 00 21 58 B0 11
E010 00 1B 01 04 00 CD 5C 00
E018 C9 00 00 00 00 00 00 00
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ?
```

(B0 ← E0を間違ってB0と入力)

入力ミスがないかどうかを確かめるためには、53頁のプログラムと画面に表示させたものをつきあわせていきます。すると、2行目の最後からひとつまえのＥ００Ｅ番地のマシン語がＥ０でなければならないのに、Ｂ０になっていることがわかります。Ｂ０が記憶されているメモリ番地が何番地のメモリであるかわかるときは、Ｓコマンドと、そのメモリ番地Ｅ００Ｅを入力すればよいのですが、わからないときは、プログラムに示されているメモリ番地を入力します。

たとえば、プログラムにつぎのように示されている場合、

E00C 21 58 E0

つぎのようにプログラムに示されているメモリ番地Ｅ００Ｃを入力します。

```
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e00c
```

(e00c ← プログラムに示されているメモリ番地を入力)

ＥＯＯＣを入力して⏎キーを叩くと、ＥＯＯＣ番地のメモリの内容２１と－を表示します。

```
コマンド ノ ニュウリョク? s
メモリバンチ ノ ニュウリョク? eOOc
EOOC 21-
```

EOOC番地のマシン語を表示したらスペースキーを叩く

ＥＯＯＣ番地のメモリの内容２１は訂正する必要がありませんから、スペースキーを叩きます。すると、つぎのように、ＥＯＯＣ番地のつぎのメモリ番地の内容５８と－を表示します。

```
コマンド ノ ニュウリョク? s
メモリバンチ ノ ニュウリョク? eOOc
EOOC 21-  58-
```

訂正する必要のないところはスペースキーを叩く

この５８も訂正する必要がありませんから、スペースキーを叩きます。すると、つぎのように訂正するＢ０が表示されますから、正しいマシン語Ｅ０を入力します。

```
コマンド ノ ニュウリョク? s
メモリバンチ ノ ニュウリョク? eOOc
EOOC 21-  58-  BO-eO
```

スペースキーを叩いて送る　BOが表示されたらEOを入力して訂正

このように、ともかく自分にわかるメモリ番地を入力して、訂正する必要のないところはスペースキーを叩いて、訂正するところまでもってきて訂正すればよいのです。

ところでマシン語のプログラムは、長くなるほど入力ミスが多くなります。また入力ミスを捜すのも大変になります。そこで入力ミスを防ぐ方法としてマシン語では、**「チェックサム」**という方法を取っています。「チェックサム」については、123頁を参照してください。

ビットとバイト

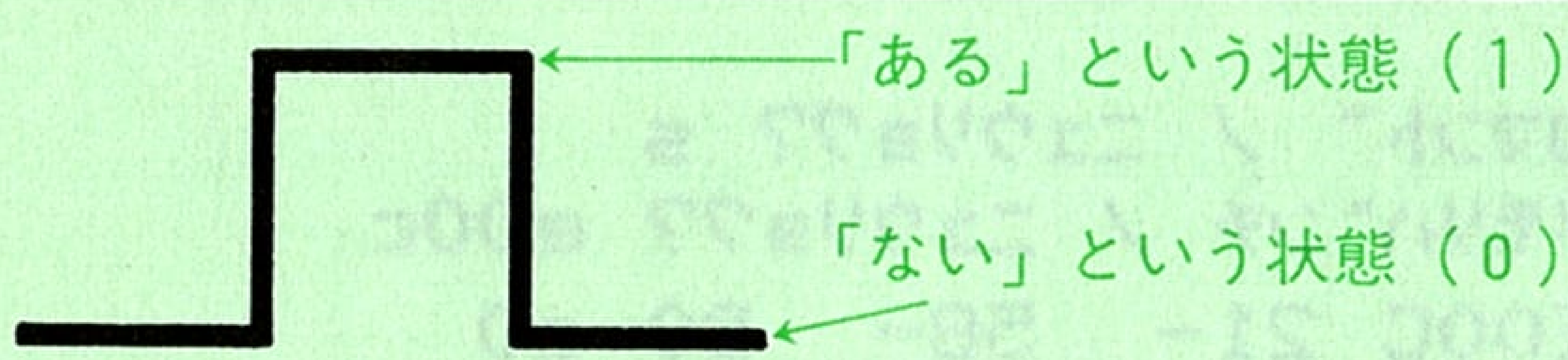
MSXパソコンのなかは、1と0だけ

マシン語は、その名前のとおり機械の言葉です。

私たちが書き表わすマシン語は、

```
3E 85       ←Aレジスタに●のキャラクタコード85Hを入れよ
CD A2 00    ←00A2番地のルーチンにジャンプせよ
C9          ←戻れ
```

というように16進数2桁の数字ですが、MSXパソコンのなかで使われている言葉は、電気信号です。電気信号といっても、MSXパソコンはデジタル回路でできていますから、デジタル信号ということになります。デジタル信号は、つぎの図に示すように「ある」か「ない」かのふたつしかありません。



「ある」という状態は、電圧がある一定の高さになった状態です。

「ない」という状態は、電圧が低くなった状態です。

したがって、MSXパソコンにわかる言葉は、「ある」とか「ない」とかのふたつだけということになります。ただ「ある」とか「ない」とかでは、私たちがプログラムを作るときに困ります。そこで、「ある」という状態に「1」を、「ない」という状態に「0」をあてはめたわけです。そして、この言葉は、1と0のふたつで数を表わすところから、**2進数**と呼ぶことにしたのです。

とすると、マシン語とは2進数で書き表わすもの、ということになりますが、16頁で説明したように、1と0で書き表わしたのでは、マシン語でプログラムを作ることが大変な作業になるので、2進数よりまだましな16進数2桁で書き表わすようにしたのです。

したがって、58頁に示した16進数2桁で書き表わしたマシン語は、キーボードのキーを叩くと同時に、つぎに示すように、1と0の8個の並びに変換されて、MSXパソコンの本体のなかに送り込まれていくことになります。

マシン語		2進数
3E	→	00111110
85	→	10000101
CD	→	11001101
A2	→	10100010
00	→	00000000
C9	→	11001001

なぜマシン語の16進数2桁が、このような2進数に変換されるのかは、102頁の16進数↔2進数変換の項を参照してください。

この1と0の8個の並びを、8ビットの2進数といいます。では、ビットとはどういうことなのか、つぎに説明することにします。

ビットとは、2進数の桁を表わす

ビットは、bitと書きます。ビットとはbinary digit（バイナリー・ディジット）の略です。binary（バイナリー）とは「2進数の」、digit（ディジット）は「桁」ということです。つまり、ビットとは「2進数の桁」ということになります。いいかえればビットとは、2進数の桁を表わす言葉ということです。

ビット（bit）＝2進数の桁を表わす言葉

たとえば2進数の1とか0は、1か0の1個で表わされていますから1桁です。桁＝ビットですから、1ビットということになります。

1
0 } 1桁で表わされているので1ビット

2進数の１０とか００は、1または0の2個で表わされていますから、2桁です。桁＝ビットですから、2ビットということになります。

10
00 } 2桁で表わされているので2ビット

2進数の１００とか１０１は、1または0の3個、つまり3桁で表わされているので、3ビットということになります。

100
101 } 3桁で表わされているので3ビット

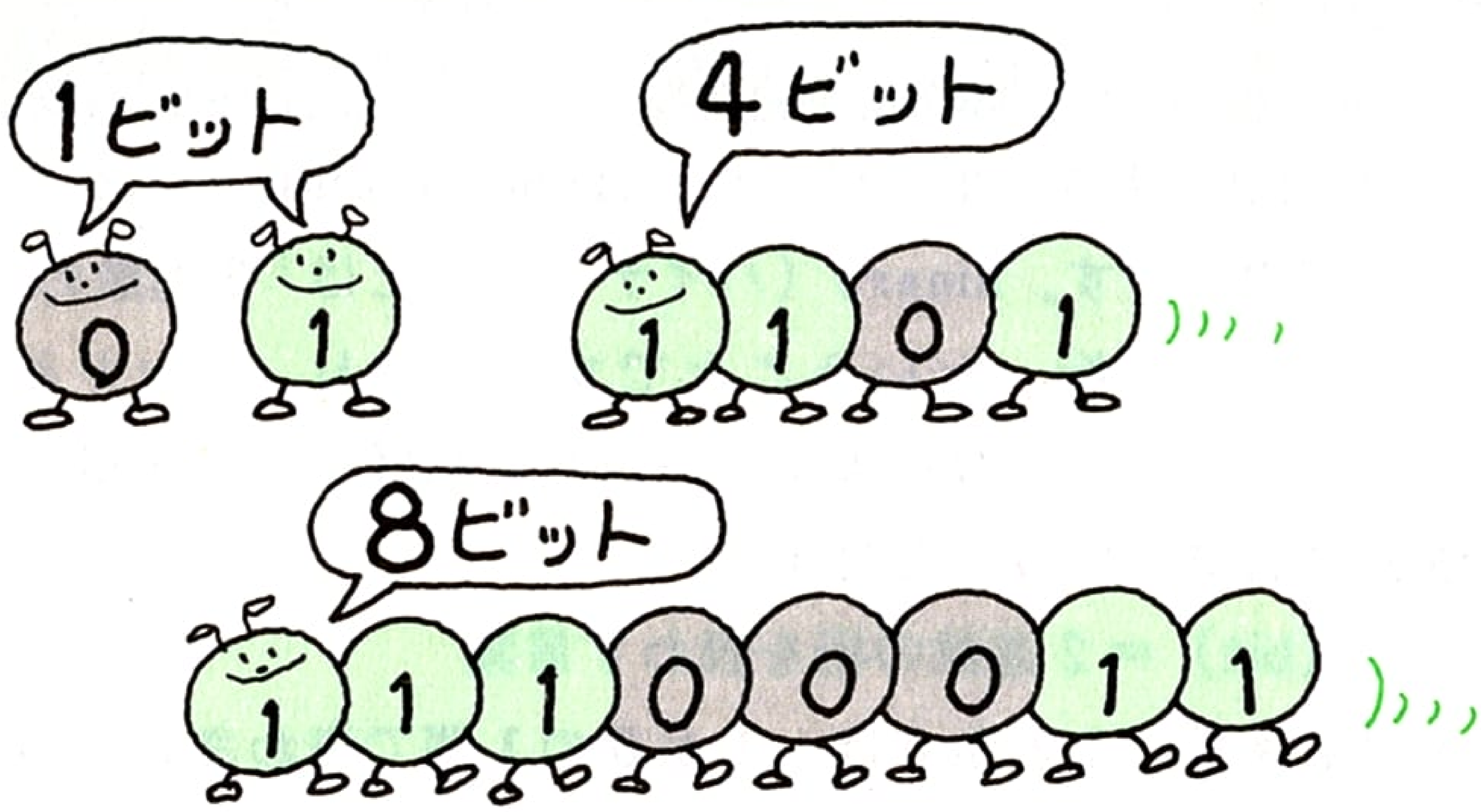
2進数の１０１０とか１１１１は、1または0の4個、つまり4桁で表わされているので、4ビットということになります。

1010
1111 } 4桁で表わされているので4ビット

2進数の００００００００とか１０１１１１００は、1または0の8個、つまり8桁で表わされているので、8ビットということになります。

00000000
10111100 } 8桁で表わされているので8ビット

このように、ビットは2進数の桁を表わしています。



バイトとは

ビットが理解できたところで、バイトです。

バイトは、byteと書きます。バイトとは、1と0の8桁、つまり、8ビットをひとつのまとまりとした表わし方です。

したがって、1バイトとは、1と0の8桁（8ビット）1個のことをいいます。

00111110←1と0の8桁（8ビット）1個が1バイト

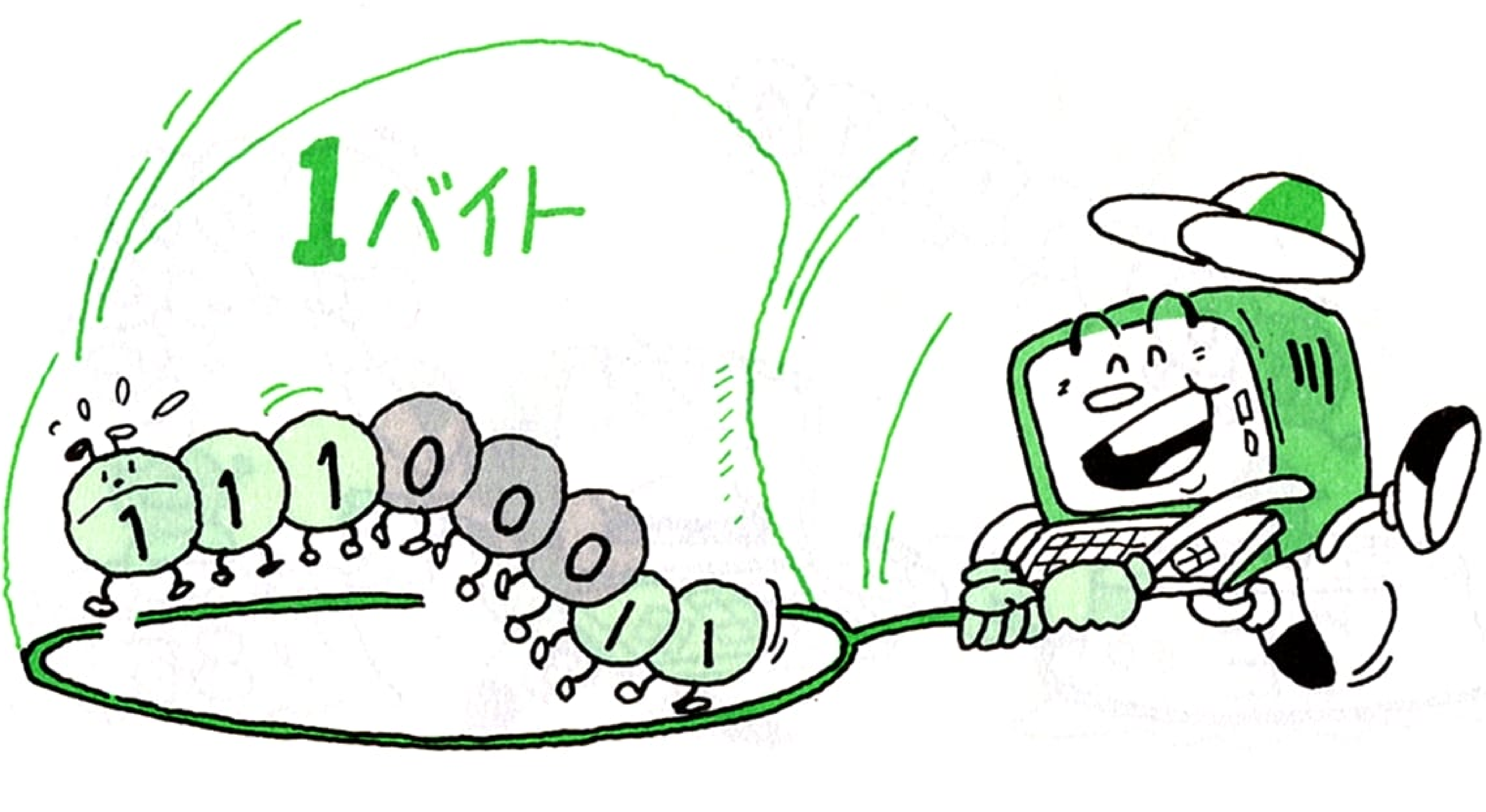
2バイトとは、1と0の8桁（8ビット）2個のことをいいます。

00111110
00001101 } 1と0の8桁（8ビット）2個が2バイト

3バイトとは、1と0の8桁（8ビット）3個のことをいいます。

00111110
00001101 } 1と0の8桁（8ビット）3個が3バイト
11001101

1と0の8桁（8ビット）をひとつのまとまりとして、バイトでいい表わすのは、わたしたちが12個をひとつのまとまりとしてダースで表わしたり、12ダースをひとつのまとまりとしてグロスで表わしたりするのとおなじです。



16進数２桁は１バイト

59頁で16進数２桁のマシン語は、キーボードのキーを叩くと同時に、８桁の２進数、つまり８ビットの２進数に変換されて、ＭＳＸパソコンの本体のなかに送り込まれると説明しました。

16進数		2 進数
3E	→	00111110
85	→	10000101

このように16進数２桁のひとつひとつは、それぞれ８ビットの２進数に変換できますから、16進数２桁の１個を１バイトといいます。

３Ｅ ←――― 16進数２桁１個は１バイト

16進数２桁が２個であれば、２バイトといいます。

３Ｅ　８５ ←――― 16進数２桁２個は２バイト

16進数２桁が３個であれば、３バイトといいます。

ＣＤ　Ａ２　００ ←――― 16進数２桁３個は３バイト

このように、16進数２桁の個数によって、２バイト、４バイト、８バイトというようないい表わし方をします。



2進数とは

2進数の数の表わし方

58頁で説明したように、ＭＳＸパソコンのなかはデジタル信号が行き交っています。デジタル信号の「ある」という状態に「1」、「ない」という状態に「0」をあてはめて表わすようにしていますから、いいかえれば、ＭＳＸパソコンのなかは「1」と「0」が行き交っているということができます。もちろん、1と0がひとつずつ行き交っているわけではありません。ＭＳＸパソコンは8ビット型です。8ビット型は、1と0の8個の並びをひとつのまとまりとして、ＣＰＵとＩ／Ｏポート、ＣＰＵとメモリの間でやりとりしていますから、

```
00111110
00100000
00000110
00100000
10000000
00110010
00010000
11100000
11001001
```

1と0の8個の並び(8ビット)をひとつのまとまりとしてやりとりしている

うえに示したように、1と0の8個の並びで行き交っているといえます。1と0の8個の並びのことを、**8ビットの2進数**といいます(58頁参照)。

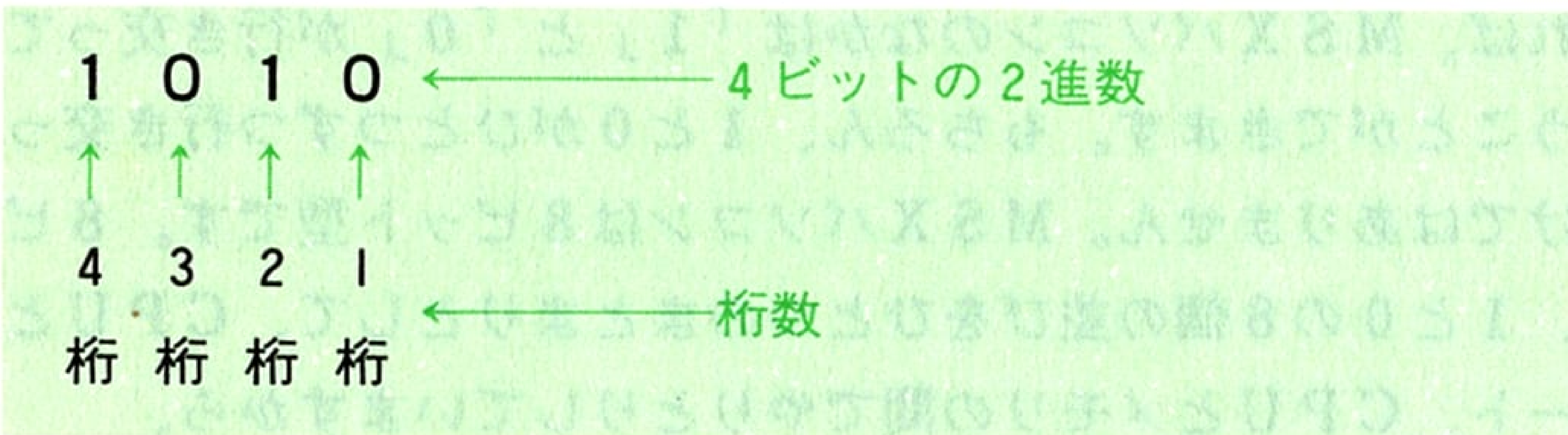
さて、これらの8ビットの2進数は、どんな数を表わしているのでしょうか。2進数をむずかしいとか、わからないという場合、たいて

い2進数の数の表わし方がわからない、ということに原因があるようですから、2進数の数の表わし方を説明します。2進数の数の表わし方が理解できると、2進数もむずかしいものではなくなると思います。

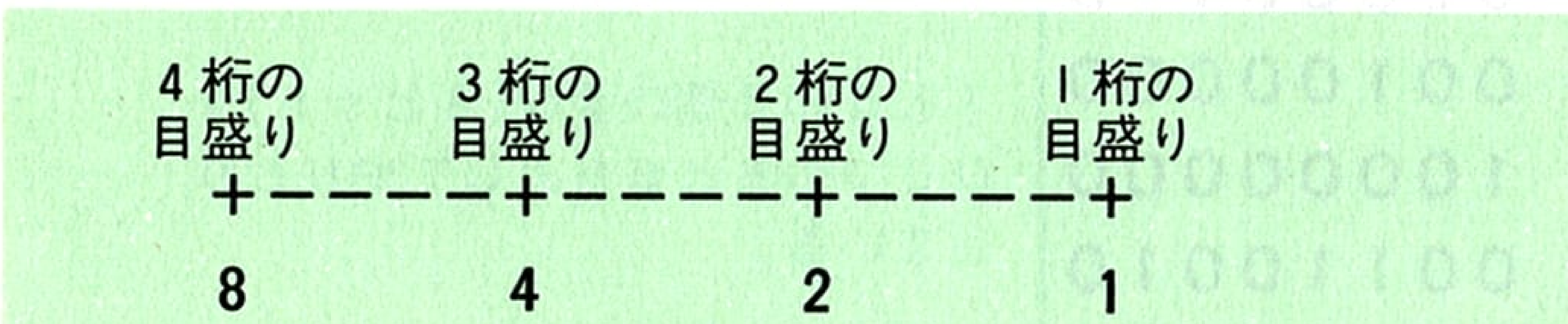
4ビットの2進数は1、2、4、8で表わす

さきに説明したようにMSXパソコンのなかでは、8ビットの2進数をひとつのまとまりとして扱っていますが、わかりやすいように、まず4ビットの2進数で、数の表わし方を説明することにします。

4ビットの2進数は、60頁で説明しているように、1と0の4桁です。桁は、右から1桁、2桁、3桁、4桁となります。

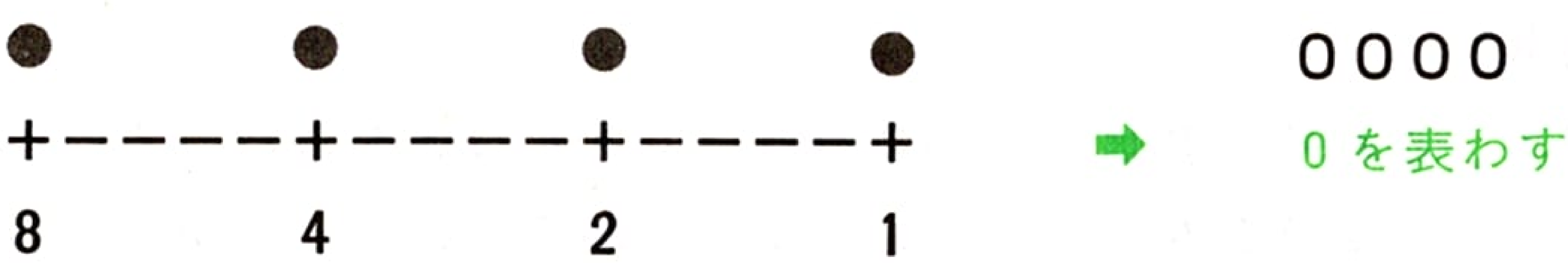


そしてつぎの図に示すように、それぞれの桁を目盛りにして、1桁の目盛りには「1」、2桁の目盛りには「2」、3桁の目盛りには「4」、4桁の目盛りには「8」という値をつけます。この値は10進数です。



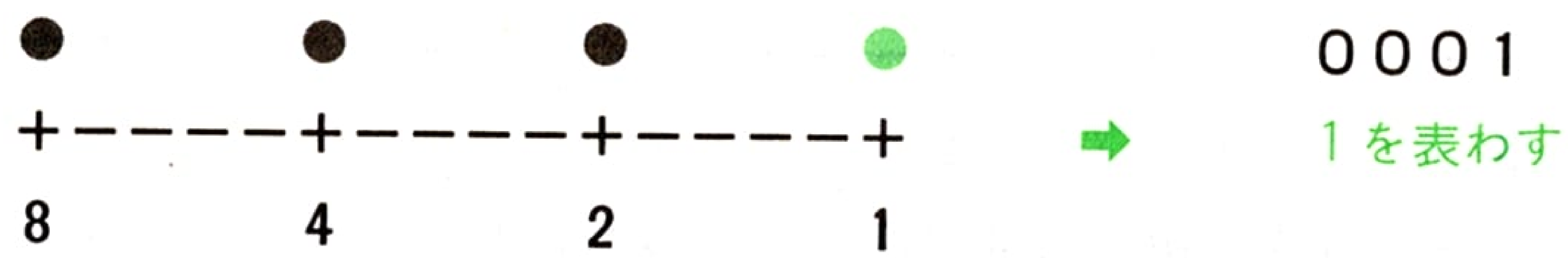
さて「ある」という状態は、電気がついている状態「オン」として●、「ない」という状態は、電気がついていない状態「オフ」として●で示すことにします。

0という数は、どこにも電気がついていない状態ですから、どの目盛りにも電気がつきません。



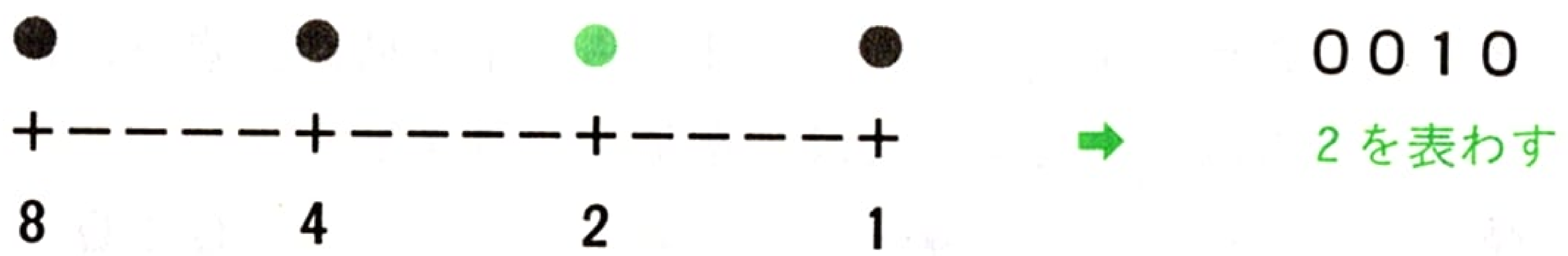
これに1か0をあてはめます。●に1、●に0です。この場合は、全部が●ですから、矢印の右に示すように、0000となります。これが、4ビットの2進数の0の表わし方です。

1という数は、1の値の目盛りに電気がつきます。そのほかの目盛りには電気がつきませんから、つぎのようになります。



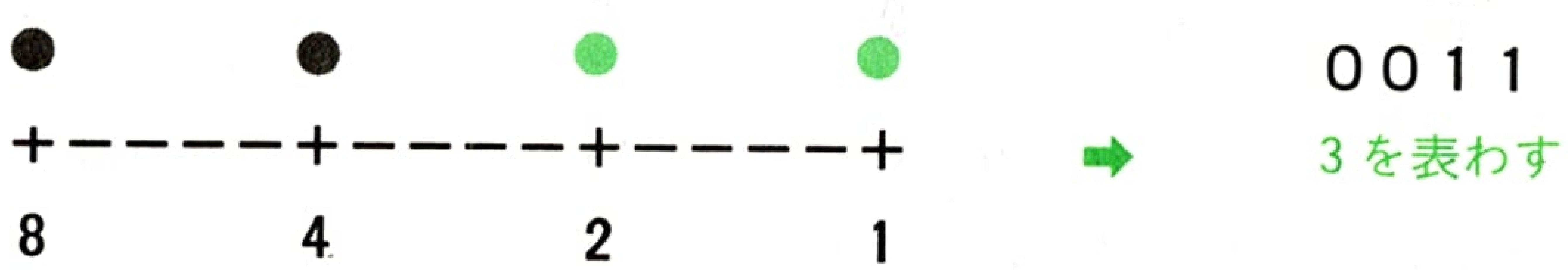
●には1、●には0をあてはめますから、1の値の目盛りだけが1で、あとは0になります。したがって、矢印の右に示すように、0001となります。これが、4ビットの2進数の1の表わし方です。

2という数は、2の値の目盛りに電気がつきます。ほかの目盛りには電気がつきません。



この場合は、2の値の目盛りが1で、ほかの目盛りは0ですから、矢印の右に示したように、0010となります。これが、4ビットの2進数の2の表わし方です。

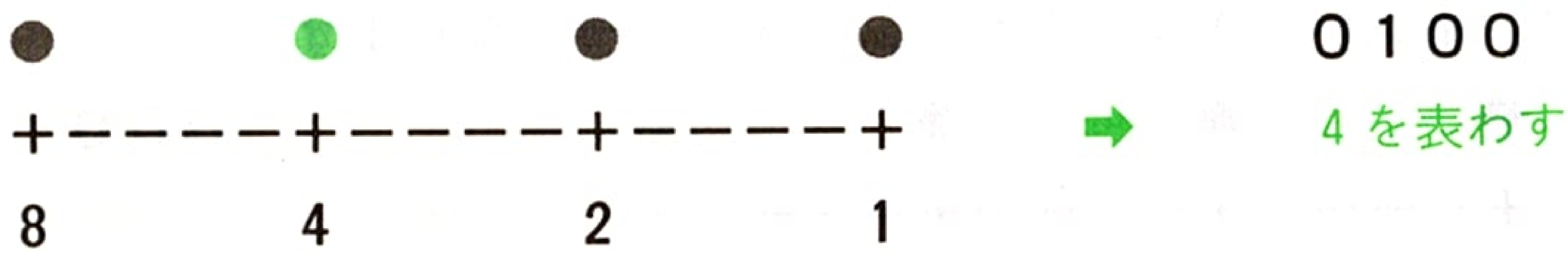
さて、3という数です。1あるいは2という数の場合は、その値がふられている目盛りがありましたが、3という値がふられている目盛りはありません。このような場合は、3という数に最も近い値で、3という数より小さい値の目盛りに、まず電気がつきます。4の値は、3より大きくなってしまいますから、2の値の目盛りに電気がつきます。そして、3から2を引きます。残りは1ですから、1の値の目盛りに電気がつきます。そのほかの目盛りには電気がつきません。



この場合は、1の値の目盛りと2の値の目盛りが1で、そのほかは

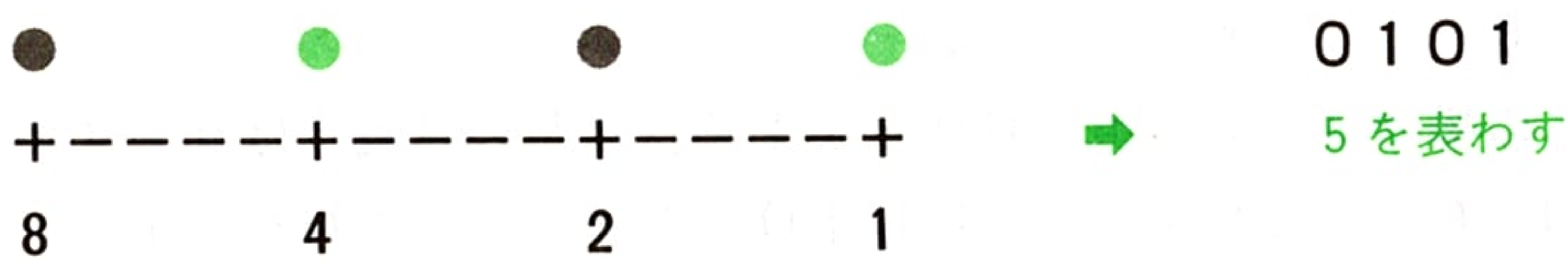
0ですから、矢印の右に示したように、００１１となります。これが、4ビットの2進数の3の表わし方です。

4という数は、4の値の目盛りに電気がつきます。そのほかの目盛りには電気がつきません。



この場合は4の値の目盛りが1で、ほかは0ですから、矢印の右に示したように、０１００となります。これが、4ビットの2進数の4の表わし方です。

5という数は、3の数とおなじように、5の値の目盛りがありません。したがって、3の数とおなじ方法で電気がつきます。5という数より小さくて、5に最も近い値は4ですから、4の値の目盛りに電気がつきます。5から4を引くと1ですから、1の値の目盛りに電気がつきます。そのほかの目盛りには、電気がつきません。



この場合は4の値の目盛りと1の値の目盛りが1で、ほかの目盛りは0ですから、矢印の右に示したように、０１０１となります。これが、4ビットの2進数の5の表わし方です。

これまでの説明で、電気のつき方、1と0のあてはめ方、つまり、4ビットの2進数の数の表わし方が理解できたことと思います。

4ビットの2進数の表わし方は、

```
+-----+-----+-----+
8     4     2     1
```

1、2、4、8の桁の値にもとづいているので、この桁の値をしっかりとおぼえておけばよいのです。

たとえば2進数の１０１０が、10進数でどんな数を示しているのか

見ただけではわからなくとも、１０１０を図に示した目盛りのうえに置いて、1が置かれた目盛りの桁の値だけを足せばよいのです。

```
1      0      1      0     1が置かれた目盛りの
+------+------+------+     値だけを足す
8      4      2      1
```

したがって、足し算はつぎのようになりますから、

８＋２＝１０

2進数の１０１０は、10進数で10を表わしている、ということがわかります。

なお、4ビットの2進数で表わすことができる数は、10進数の0から15までです。つぎに10進数の0から15までの1、2、4、8の桁の値への電気のつき方と、4ビットの2進数を表にまとめて示します。

8－－4－－2－－1	2進数	10進数
● ● ● ●	０００００	０
● ● ● ○	０００１	１
● ● ○ ●	００１０	２
● ● ○ ○	００１１	３
● ○ ● ●	０１００	４
● ○ ● ○	０１０１	５
● ○ ○ ●	０１１０	６
● ○ ○ ○	０１１１	７
○ ● ● ●	１０００	８
○ ● ● ○	１００１	９
○ ● ○ ●	１０１０	１０
○ ● ○ ○	１０１１	１１
○ ○ ● ●	１１００	１２
○ ○ ● ○	１１０１	１３
○ ○ ○ ●	１１１０	１４
○ ○ ○ ○	１１１１	１５

8ビットは、16、32、64、128の桁の値を追加

つぎは、8ビットの2進数です。8ビットの2進数は、

0	0	1	1	1	1	1	0	←8ビットの2進数
↑	↑	↑	↑	↑	↑	↑	↑	
8桁	7桁	6桁	5桁	4桁	3桁	2桁	1桁	←桁数

8桁ですから、4ビットの2進数の1から4までの桁に、5から8までの桁を追加することになります。追加した桁には、10進数で16、32、64、128の桁の値をつけます。

```
+----+----+----+----+----+----+----+
128  64   32   16   8    4    2    1
```

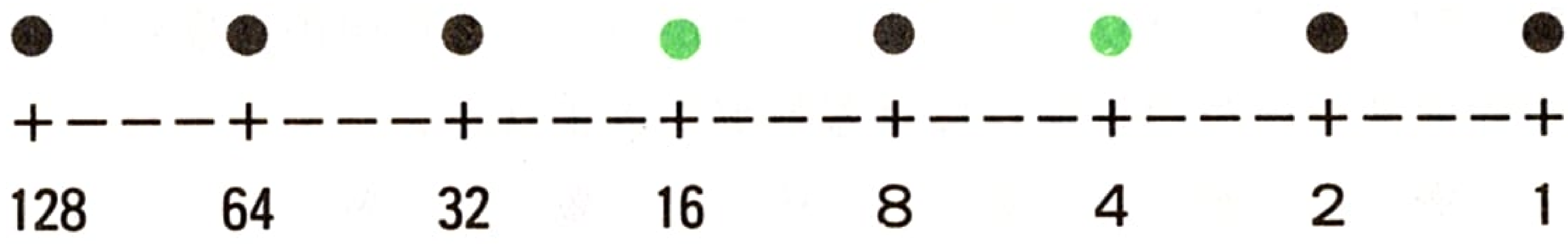
目盛りへの電気のつき方は、4ビットのときとおなじです。つまり表わす数に応じた値を目盛りの値のなかから選びだして、その値の目盛りに電気をつければよいのです。そして、電気がついた目盛りは1、電気がつかない目盛りは0とします。

ところで、8ビットの場合の0から15までの数の表わし方は、つぎのように、4ビットで表わした数のまえに0を4個つけるだけです。

4ビットの2進数の1		8ビットの2進数の1
0001	➡	00000001
4ビットの2進数の5		8ビットの2進数の5
0101	➡	00000101

したがって、ここでは、4ビットで表わすことができる15までは省略して、16以上の数を順を追わないで取りあげていくことにします。

まず、20という数です。20という値は目盛りにはありませんから、20という数より小さくて20に最も近い値を、目盛りの値のなかから選びます。すると16ですから、16の値の目盛りに電気がつきます。20から16を引くと4です。そこで、4の値の目盛りに電気がつきます。そのほかの目盛りには電気がつきません。



●に1、●に0をあてはめると、

００010100←――20を表わす

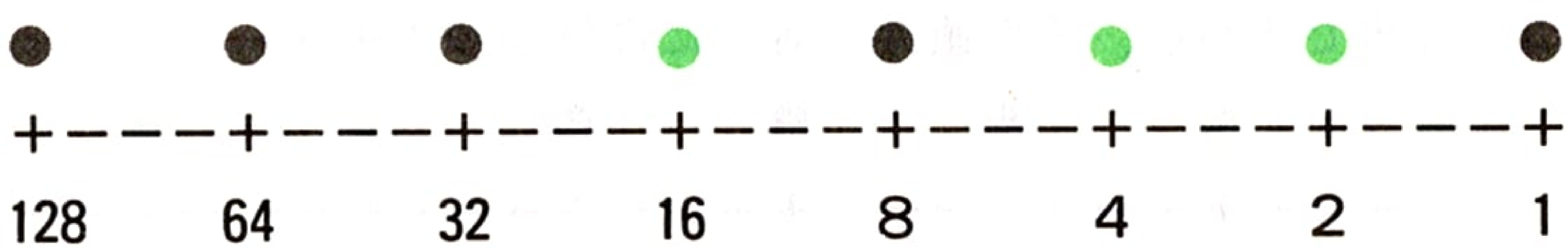
となります。これが、8ビットの2進数の20の表わし方です。

22という数は、20という数のときとおなじように、16の値の目盛りと4の値の目盛りに電気がつきます。

22－16＝6

6－4＝2

で、まだ2が残りますから、さらに2の値の目盛りに電気がつきます。そのほかの目盛りには電気がつきません。

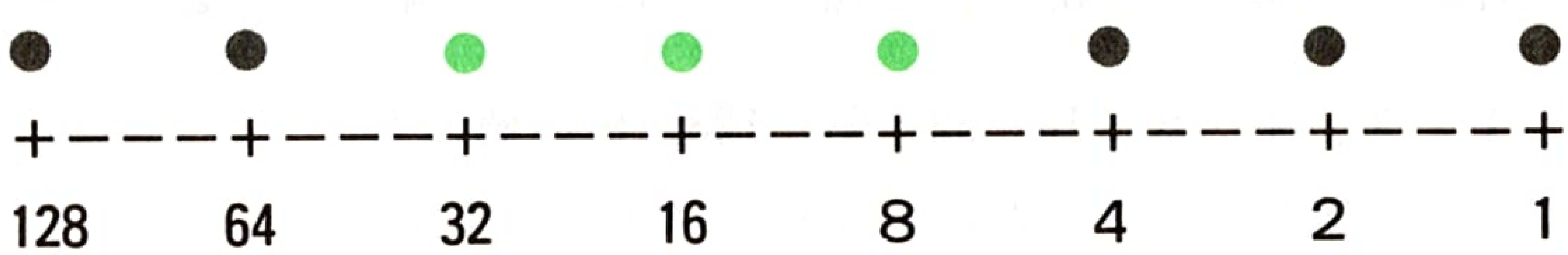


●に1、●に0をあてはめると、

00010110←――22を表わす

となります。これが、8ビットの2進数の22の表わし方です。

56という数は、まず32の値の目盛りに電気がつきます。56－32＝24で、残りは24ですから、つぎに16の値の目盛りに電気がつきます。24－16＝8で、残りは8ですから、8の目盛りに電気がつきます。ほかの目盛りには、電気がつきません。



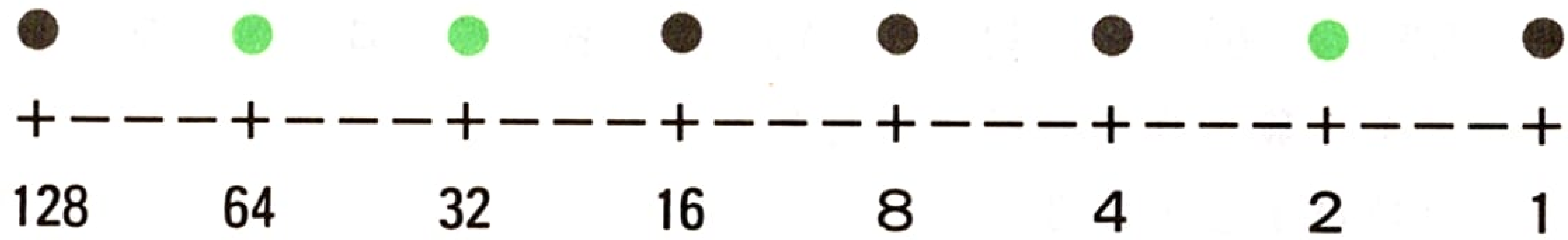
●に1、●に0をあてはめると、

00111000←――56を表わす

となります。これが8ビットの2進数の56の表わし方です。

98という数は、64の値の目盛りに電気がつきます。98－64＝34で、残りは34です。したがって、つぎに32の値の目盛りに電気がつきます。

34－32＝2で、残りは2です。したがって、2の値の目盛りに電気がつきます。ほかの目盛りには電気がつきません。

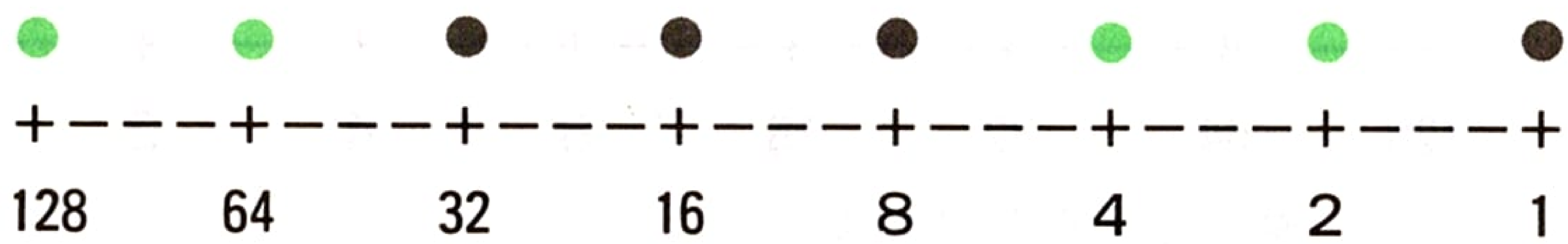


●に1、●に0をあてはめると、

01100010 ←—— 98の表わし方

となります。これが、8ビットの2進数の98の表わし方です。

では最後は、198という数です。198という数は、最も大きい128の値の目盛りに電気がつきます。198－128＝70で、残りは70です。したがって、つぎに64の値の目盛りに電気がつきます。70－64＝6で、残りは6です。6という数は、4の値の目盛りと2の値の目盛りに電気がつけば6ですから、その値の目盛りに電気がつきます。

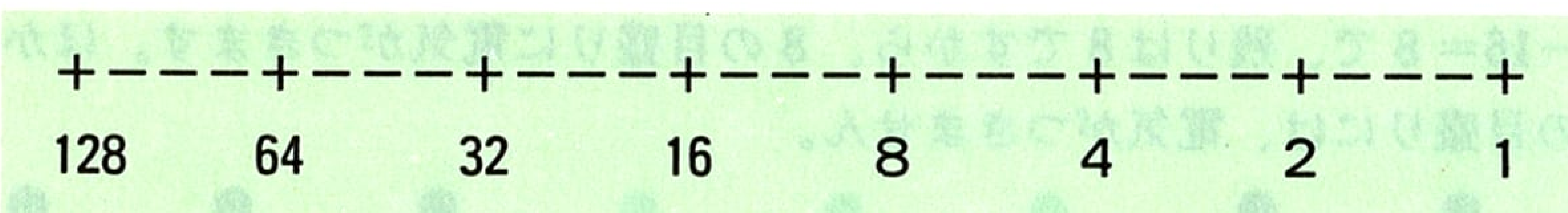


●に1、●に0をあてはめると、

11000110 ←—— 198の表わし方

となります。これが、8ビット2進数の198の表わし方です。

8ビットの2進数の数の表わし方は、



1、2、4、8、16、32、64、128の桁の値にもとづいているので、この桁の値をしっかりおぼえておけばよいのです。

この桁の値をおぼえておくと、最初に示した8ビットの2進数が、

00111110 ←—— 10進数で62

00100000 ←—— 10進数で32

10進数で62、10進数で32を表わしているといわれても、つぎのようにすれば、すぐ納得がいくはずです。

たとえば、00111110の場合は、

```
 0    0    1    1    1    1    1    0
 +----+----+----+----+----+----+----+
128   64   32   16   8    4    2    1
```

こうなりますから、1が置かれている目盛りの値だけを足します。

32＋16＋8＋4＋2＝62

したがって8ビットの2進数00111110は、10進数で62ということになります。この目盛りにつけられた桁の値を知らないと、どうして00111110が10進数で62なのかわからない、ということになるわけです。

もちろん、目盛りにつけられた桁の値を知っていると、8ビットの2進数が、どんな値を表わしているかも読み取ることができます。

たとえば、8ビットの2進数の11001001は、

```
 1    1    0    0    1    0    0    1
 +----+----+----+----+----+----+----+
128   64   32   16   8    4    2    1
```

ですから、まえの例のように、1が置かれている目盛りの値だけを足していきます。

128＋64＋8＋1＝201

したがって8ビットの2進数11001001は、10進数で201である、ということがわかります。

最初に説明した4ビットの2進数で表わすことができる数は、0から15まででしたが、8ビットの2進数で表わすことができる数は、0から255までとなります。

なお82頁の2進数↔10進数変換では、もっと簡単に4ビットの2進数あるいは8ビットの2進数が、10進数でどんな値を表わしているかがわかる方法を説明します。簡単にわかる方法があれば、それを使ったほうがよいに決まっていますが、簡単な方法は、2進数を10進数に機械的に変換するだけですから、なぜそのような数になるのか、その理由がわかりません。理由を知ったうえで、それを使うことが大切なので、ここで説明したことは忘れずにおぼえておいてください。

63頁に示した8ビットの2進数の意味

では、63頁に示した8ビットの2進数の意味について説明しておくことにします。63頁に示した8ビットの2進数を、10進数に変換しても意味はわかりませんから、16進数に変換します。なぜなら、マシン語は16進数2桁で書き表わすからです。8ビットの2進数を、16進数になおすことはむずかしいことではありません。2進数を、16進数に変換する方法を知っていると簡単にできます。2進数↔16進数変換については、98頁を参照してください。

00111110	⟹	3E
00100000	⟹	20
00000110	⟹	06
00100000	⟹	20
10000000	⟹	80
00110010	⟹	32
00010000	⟹	10
11100000	⟹	E0
11001001	⟹	C9

つぎに、マシン語をアセンブリ言語で示します。本当はアセンブリ言語で書いてからマシン語になおします。アセンブリ言語をマシン語になおす方法については、110頁を参照してください。

メモリ番地	マ シ ン 語	アセンブリ言語
E000	3E 20	LD A, 20H
E002	06 20	LD B, 20H
E004	80	ADD A, B
E005	32 10 E0	LD (E010H), A
E008	C9	RET

このマシン語のプログラムは、足し算のプログラムです。

実行させるときは、プログラムにつけられているＥ０００番地から入力して、Ｇコマンドで実行させます。さて、１行目です。

　　ＬＤ　Ａ，２０Ｈ←――Aレジスタに20Hを入れよ

ＬＤはロード命令で、「入れよ」という命令です。ＡはＡレジスタです。２０Ｈは数値（10進数で32）です。したがって、Ａレジスタに２０Ｈを入れよということになります。つぎは、２行目です。

　　ＬＤ　Ｂ，２０Ｈ←――Bレジスタに20Hを入れよ

ＬＤは命令で、１行目とおなじです。ＢはＢレジスタです。２０Ｈは１行目とおなじで数値です。したがって、Ｂレジスタに２０Ｈを入れよということになります。

このようにレジスタに、16進数２桁（８ビット）の値を入れることを、

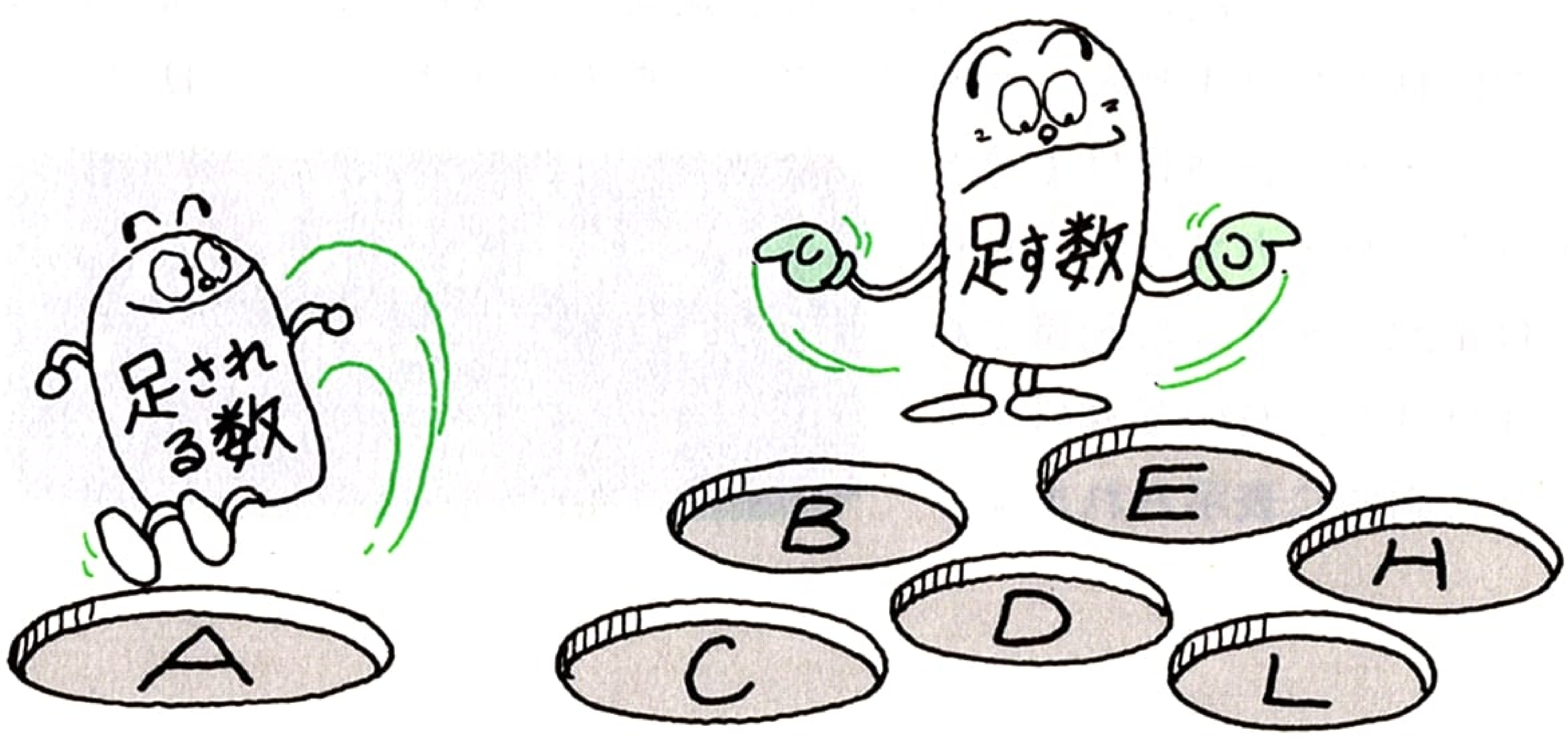
８ビット・ロード命令＝16進数２桁の値をレジスタに入れる

８ビット・ロード命令といいます。つぎは、３行目です。

　　ＡＤＤ　Ａ，Ｂ←――Aレジスタの内容にBレジスタの内容を足せ

ＡＤＤは命令です。add（アド）は英語で、「足す」とか「加える」という意味です。ＡはＡレジスタ、ＢはＢレジスタです。したがって、Ａレジスタの内容にＢレジスタの内容を足せ、ということになります。

足し算の場合、足される数は、Ａレジスタに入れると決まっていますが、足す数は、Ｂ、Ｃ、Ｄ、Ｅ、Ｈ、Ｌレジスタのいずれかに入れます。レジスタについては、144頁を参照してください。



この場合、Ａレジスタの内容は２０Ｈ、Ｂレジスタの内容も２０Ｈですから、ＡＤＤ（アド）命令で、

２０Ｈ＋２０Ｈ＝４０Ｈ（10進数で64）

が行われます。足し算の結果の４０Ｈは、Ａレジスタに残ります。

４０Ｈ　➡　Ａレジスタ

このように、16進数２桁の数値の足し算の場合は、Ａレジスタが中心になって行われます。16進数２桁の数値の足し算のことを、

８ビット算術演算命令＝16進数２桁の値の足し算

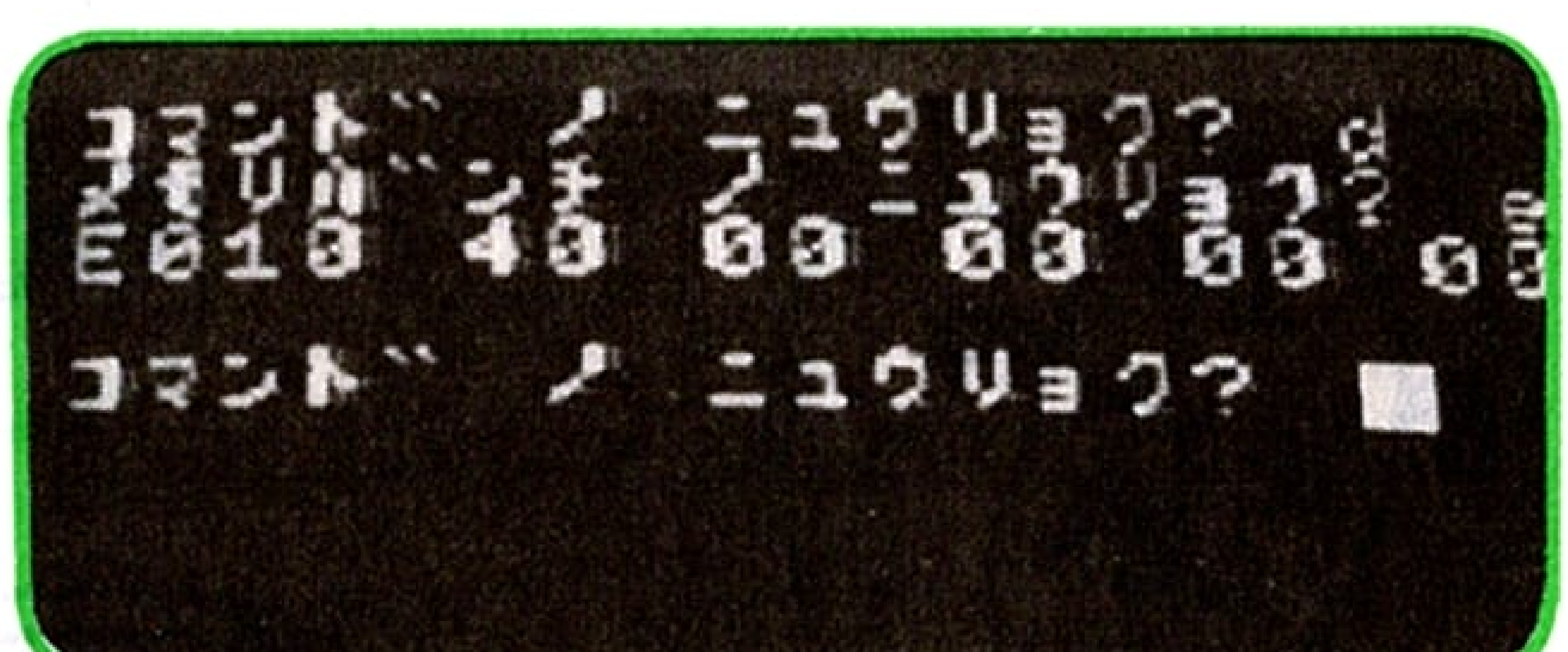
８ビット算術演算命令といいます。つぎは、４行目です。

ＬＤ　（Ｅ０１０Ｈ），Ａ ←――Ａレジスタの内容をＥ０１０番地に記憶

ＬＤは、１行目、２行目とおなじ命令です。ＬＤ命令のあとに、カッコで括られて４桁の16進数があるときは、その数値はメモリ番地と決まっています。この場合は、Ｅ０１０番地のメモリということです。ＡはＡレジスタです。したがって、Ａレジスタの内容をＥ０１０番地のメモリに入れよ、ということになります。Ａレジスタには３行目のＡＤＤ命令で、足し算された結果の４０Ｈが記憶されていますから、その４０ＨがＥ０１０番地のメモリに記憶されます。最後の行の、

ＲＥＴ ←――マシン語モニタに戻れ

は、RETURN（リターン）命令です。マシン語のプログラムを実行したあと、このＲＥＴ命令でマシン語モニタに戻ります。ＭＳＸパソコンのマシン語のプログラムでは、最後にかならずＲＥＴ命令を置かなければなりません。なおプログラムを実行させたあと、Ｄコマンドと、メモリ番地Ｅ０１０とＥ０１７を入力すると、Ｅ０１０番地のメモリに記憶された４０Ｈが、右の写真に示すように画面に表示されます。



16進数とは

16進数は、16で桁あがりする数

私たちはふつう、ベーシックでプログラムを組んでいます。ベーシックの場合、私たちがふだん使っている10進数を使います。ところが、マシン語でプログラムを組むときは、10進数はまったく使いません。マシン語そのものが16進数ですし、マシン語を記憶させるメモリ番地も16進数、数値も16進数というように、すべてが16進数となります。

ところで10進数と16進数とでは、どこが違うのでしょうか。私たちがふだん使っている10進数は、0から9まで数えてきて、10で桁あがりします。

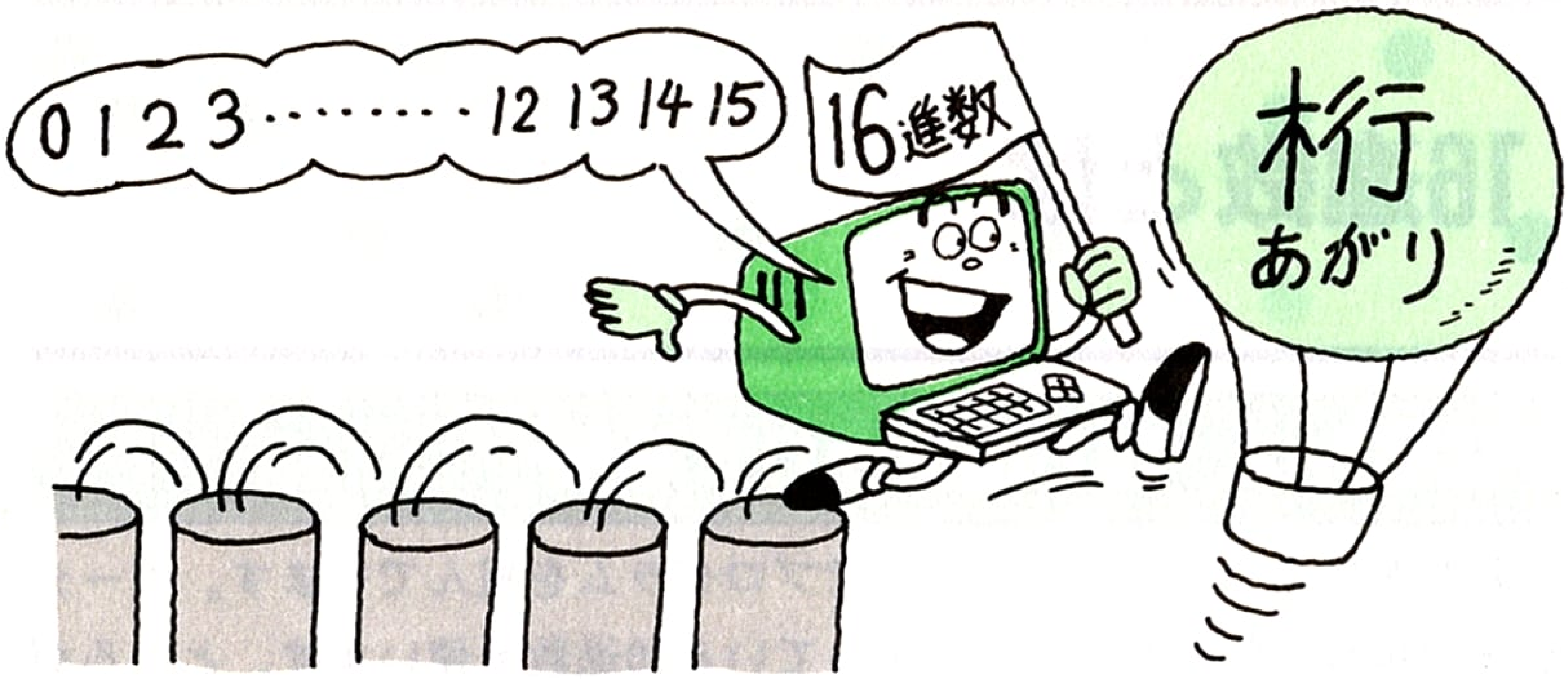
0 1 2 3 4 5 6 7 8 9 ←10進数は0から9まで数えて
10 ←— 10で桁あがりする

16進数は、0から15まで数えてきて、16で桁あがりします。

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ← 0から15まで数えてきて
16 ←— 16で桁あがりする

つまり10進数では、0から9までが1桁で、10から2桁になりますが、16進数では、0から15までが1桁で、16から2桁になります。

もちろん、こればかりではありません。0から9までの1桁の数字は10進数とおなじなので、16進数でもそのまま1桁の数字として使いますが、10、11、12、13、14、15は10進数では2桁の数字なので、16進数でそのまま1桁の数字として使うことができません。なぜなら、10進数で2桁の数字を、16進数で1桁の数字として使うと、こんがらがってしまうからです。



そこで16進数では、10から15までを1桁として表わす、新しい数字が必要になります。この場合、10から15までを1桁で表わすことができれば、どんなものでもよかったのですが、16進数では、A、B、C、D、E、Fの6個のアルファベットを使うことにしました。そのため16進数の0から15までは、つぎのように0から9までの10個の数字とAからFまでの6個のアルファベットの合計16個の数字を使って表わします。

0 1 2 3 4 5 6 7 8 9 A B C D E F

したがって、16進数で使っているA、B、C、D、E、Fは、文字どおりのエイ、ビー、シーの意味ではなくて、それぞれ10進数の10、11、12、13、14、15に対応しています。

A B C D E F ←——16進数
↕ ↕ ↕ ↕ ↕ ↕
10 11 12 13 14 15 ←——10進数

そして、つぎのように0からFまで数えてきて、

0 1 2 3 4 5 6 7 8 9 A B C D E F
10 ←——10で桁あがり

10で桁あがりします。この16進数の10は、数字どおりの10ではなくて、Fが10進数の15ですから、10進数の16にあたります。

これが、16進数の数の表わし方です。私たちがふだん使っている10進数が、10で桁あがりする数、いいかえれば、10をひとつの区切りとして表わす数であるのに対して、16進数は16で桁あがりする数、いいかえれば、16をひとつの区切りとして表わす数なのです。

16進数をマシン語で使うのは、16進数は4ビットの2進数を1桁で、8ビットの2進数を2桁で表わすことができるからです。

4ビットの2進数＝16進数1桁
8ビットの2進数＝16進数2桁

2進数と16進数がどのように対応しているかを説明するまえに、16進数2桁以降がどのように表わされているかを参考までに示しておくことにします。カッコのなかは、その16進数に対応する10進数です。

10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
(16)	(17)	(18)	(19)	(20)	(21)	(22)	(23)	(24)	(25)	(26)	(27)	(28)	(29)	(30)	(31)
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
(32)	(33)	(34)	(35)	(36)	(37)	(38)	(39)	(40)	(41)	(42)	(43)	(44)	(45)	(46)	(47)
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
(48)	(49)	(50)	(51)	(52)	(53)	(54)	(55)	(56)	(57)	(58)	(59)	(60)	(61)	(62)	(63)
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
(64)	(65)	(66)	(67)	(68)	(69)	(70)	(71)	(72)	(73)	(74)	(75)	(76)	(77)	(78)	(79)
	⋮														
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
(144)	(145)	(146)	(147)	(148)	(149)	(150)	(151)	(152)	(153)	(154)	(155)	(156)	(157)	(158)	(159)
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
(160)	(161)	(162)	(163)	(164)	(165)	(166)	(167)	(168)	(169)	(170)	(171)	(172)	(173)	(174)	(175)
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
(176)	(177)	(178)	(179)	(180)	(181)	(182)	(183)	(184)	(185)	(186)	(187)	(188)	(189)	(190)	(191)
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
(192)	(193)	(194)	(195)	(196)	(197)	(198)	(199)	(200)	(201)	(202)	(203)	(204)	(205)	(206)	(207)
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
(208)	(209)	(210)	(211)	(212)	(213)	(214)	(215)	(216)	(217)	(218)	(219)	(220)	(221)	(222)	(223)

E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
(224)	(225)	(226)	(227)	(228)	(229)	(230)	(231)	(232)	(233)	(234)	(235)	(236)	(237)	(238)	(239)
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
(240)	(241)	(242)	(243)	(244)	(245)	(246)	(247)	(248)	(249)	(250)	(251)	(252)	(253)	(254)	(255)
100	101	102	103	104	105	106	107	108	109	10A	10B	10C	10D	10E	10F
(256)	(257)	(258)	(259)	(260)	(261)	(262)	(263)	(264)	(265)	(266)	(267)	(268)	(269)	(270)	(271)

このように、16進数は、F、1F、2F、3Fというように、Fのつぎで桁あがりしていきます。したがって、カッコのなかに示すように、16進数の10は10進数の16、16進数の20は10進数の32、16進数の30は10進数の48に対応することになります。

16進数のあとにはHをつける

16進数は10進数と区別するために、数字のおわりにHをつけて表わします。このHは、Hexadecimal（ヘキサデシマル・16進数）の頭文字です。

ただし、マシン語は16進数で書き表わしますが、マシン語にはHはつけません。アセンブリ言語の場合は、数値やメモリ番地を16進数で書き表わすので、それにはHをつけます。

FFH　85H　E010H　C020H

つぎに、参考までにアセンブリ言語のプログラムを示します。

```
LD  A, 50H      ←50HをAレジスタに入れよ
LD  B, 25H      ←25HをBレジスタに入れよ
SUB  B          ←Aレジスタの50HからBレジスタの25Hを引け
LD  (E010H), A  ←引き算の結果をE010番地のメモリ番地に入れよ
RET             ←マシン語モニタに戻れ
```

1行目のＬＤ　A，５０Ｈの50は数値なので、Hがついています。５０Ｈは、10進数では80です。2行目のＬＤ　B，２５Ｈの２５Ｈは数値なので、Hがついています。２５Ｈは、10進数では37です。4行

目のＬＤ（Ｅ０１０Ｈ），ＡのＥ０１０Ｈはメモリ番地なので、Ｈがついています。このようにアセンブリ言語では、数値とかメモリ番地のとき、Ｈをつけて表わします。

4ビットの2進数は、16進数1桁で表わせる

77頁で、４ビットの２進数は16進数の１桁で、８ビットの２進数は16進数の２桁で表わすことができると説明しましたが、４ビットの２進数と16進数の１桁が基準になるので、まず、４ビットの２進数が16進数の１桁で、どのように表わすことができるかを示します。

４ビットの２進数とは、１か０のひとつが１桁ですから、

0000　　0011　　1101　　1111

というように、４桁の２進数ということになります。この４ビットの２進数で表わすことができる数は、

0000 ←——10進数で0

～

1111 ←——10進数で15

まで、つまり10進数の０から15までです。

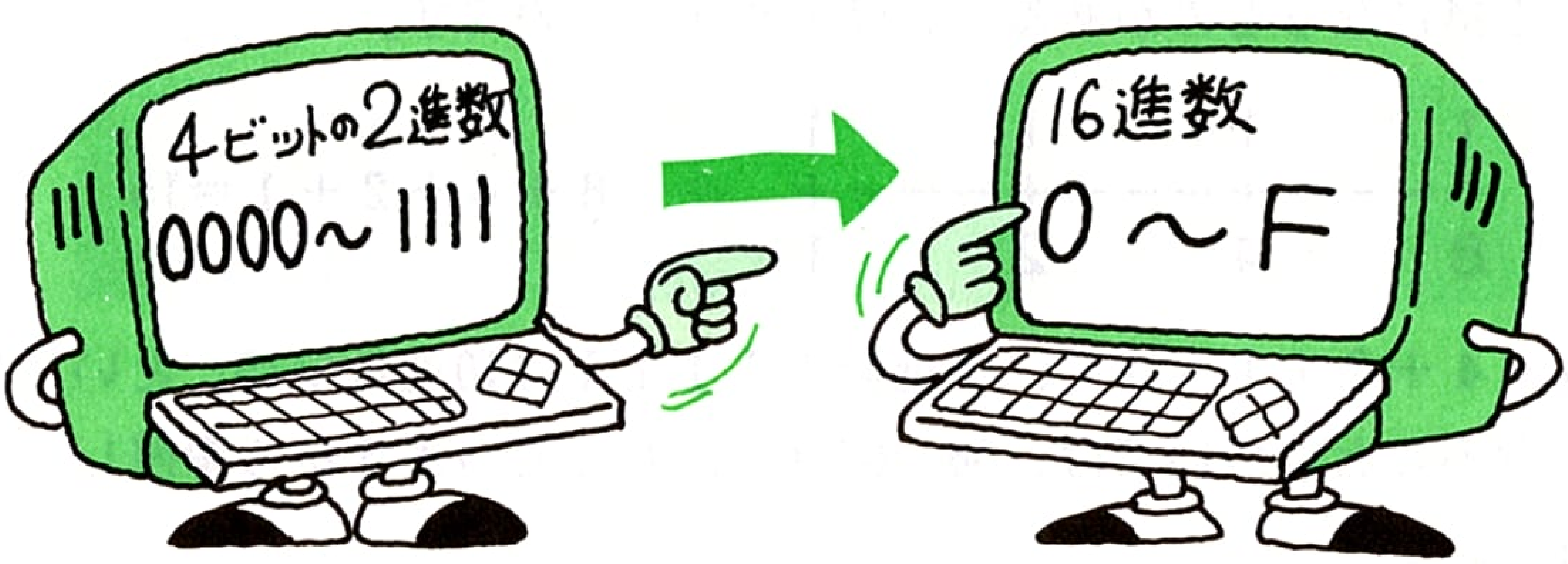
では、４ビットの２進数で表わすことができる００００から１１１１までを、16進数ではどのように表わすかというと、

0 ←——10進数で0

～

F ←——10進数で15

０からＦまでの１桁で表わします。したがって、つぎの表に示すよ



うに4ビットの2進数0000から1111までは、16進数の1桁で表わすことができる、ということになるのです。

4ビットの2進数=16進数1桁

2 進 数	16 進 数	10 進 数
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

ところで、16進数のFが10進数の15である、ということはおぼえやすいと思います。2進数の1111が10進数の15である、ということも慣れてしまえばどうということはありませんが、最初のときは、なにか根拠になるものがあるとおぼえやすいものです。

```
1       1       1       1
+-------+-------+-------+  ➡  8+4+2+1=15
8       4       2       1
```

8+4+2+1=15で、2進数の1111は10進数で15、という根拠になることについては、66〜67頁で説明していますから参照してください。

8ビットの2進数は、16進数2桁で表わせる

つぎは8ビットの2進数を、16進数2桁で表わすことにします。8ビットの2進数を16進数の2桁で表わすには、まえに説明した4ビットの2進数の組み合わせです。

8ビットの2進数は、つぎのように1と0の8桁です。

00111110　　00100000　　00000110　　11001001

この8ビットの2進数を、4ビットずつのふたつにわけます。たとえば、00111110を例にすると、つぎのように、

0 0 1 1 1 1 1 0

0 0 1 1　　1 1 1 0　　4ビットずつのふたつにわける

上位ビット　　下位ビット

ふたつにわけます。この場合、最初の4ビットを**上位ビット**、あとのほうの4ビットを**下位ビット**といいます。このように００１１と１１１０の4ビットずつにわけたら、80頁に示した4ビットの2進数と16進数の対応表を見ます。

4ビットの2進数００１１は、16進数で３です。4ビットの2進数１１１０は、16進数でＥです。この16進数の３とＥを、上位ビットと下位ビットにそれぞれあてはめます。

0 0 1 1　　1 1 1 0

3　　E

3 E H　←── 3とEをひとつにまとめる

そして、上位ビットにあてはめた３と下位ビットにあてはめたＥとを、ひとつにまとめます。まとめると３Ｅになります。したがって8ビットの2進数00111110は、16進数で３ＥＨということになりますから、8ビットの2進数は、16進数の2桁で表わすことができるのです。

8ビットの2進数＝16進数2桁

2進数を10進数に変換する

MSXベーシックの&Bを使って変換

私たちがふだん使っている数は、10進数です。したがって、1と0で表わされた2進数をみても、なにがなんだかわかりません。そこで2進数を10進数に変換する必要がおこります。

2進数を10進数に変換する最も簡単な方法は、MSXベーシックの&Bを使うことです。&Bは、&Bのあとにつづく1と0の並びが、2進数であることを示します。

&B=1と0の並びが2進数であることを示す

この&Bを使って2進数を10進数に変換するには、つぎのようにダイレクトモードで、PRINTのあとに&Bと10進数に変換する2進数を入力すればよいのです。

```
print &b10100110
 166 ←——10進数に変換されて表示
OK
```

うえの場合は、8ビットの2進数ですが、4ビットの2進数の場合もおなじです。

```
print &b1111
 15
OK
```

これがダイレクトモードで、ＭＳＸベーシックの＆Ｂを使って、２進数を10進数に変換する方法です。もちろん、ダイレクトモードでは２進数のひとつひとつしか、10進数に変換することができません。いくつもの２進数を10進数に変換するには、プログラムモードで行います。つぎに、２進数を10進数に変換するプログラムを示します。

```
10 INPUT A$
20 IF A$= "999" THEN 50
30 PRINT VAL("&b"+A$)
40 GOTO 10
50 END
```

マシン語を学ぼうとする人にとっては、このくらいのプログラムはわかると思いますが、いちおう簡単に説明しておくことにします。

行番号10のINPUT文で文字変数Ａ＄に、10進数に変換する２進数を入力します。行番号30のＶＡＬ（〝＆ｂ〃＋Ａ＄）でＡ＄に入力した２進数を数値に変換します。ＶＡＬ関数は数値に変換する関数です。そしてPRINTで、数値に変換した２進数を、10進数で画面に表示します。行番号40のＧＯＴＯ文で、繰り返えすために行番号10に戻ります。

２進数を10進数に変換することがおわったら、999を入力します。行番号20のＩＦ～THEN文で判断されて、行番号50のＥＮＤ文にジャンプしておわりとなります。

つぎに、このプログラムの実行結果を示しておきます。

```
run
? 11111111
 255
? 1111
 15
? 10101111
 175
```

以上、&Bを使って、2進数を10進数に変換する方法を説明しました。この方法は非常に簡単で、しかもMSXパソコンが変換してくれるので、これほど正確なものはありません。ただこの方法だと、2進数の10101111が、どうして10進数で175になるのかといったことがわかりません。このことがわかるように、63頁で2進数の数の表わし方を説明したのですが、ここでビット番号について知っておいてもらうために、あらためて2進数の数の表わし方について、説明しておくことにします。

2進数の各桁にはビット番号がついている

私たちが使っているMSXパソコンは、8ビット型です。したがって、8ビット(1バイト)の2進数をひとつのまとまりとして扱っています。8ビットの2進数は1と0の8個の並び、つまり8桁です。

10101111　　11111111　　00010101

この8桁の2進数には、つぎの図に示すように右端から、bit 0、bit 1、bit 2、bit 3というように番号がつけられています。これを**ビット番号**といいます。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

最上位ビット　　最下位ビット

このうちのbit 0を**最下位ビット**、bit 7を**最上位ビット**といいます。

さきに示した8ビットの2進数10101111を例にとると、つぎのようにビット番号がつけられていることになります。

1	0	1	0	1	1	1	1
↑	↑	↑	↑	↑	↑	↑	↑
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

このように8ビットの2進数には、ビット番号がつけられているということをおぼえておいてください。

さて、このbit 0 からbit 7 までに、68頁で桁の値としてつけた、1 、2 、4 、8 、16、32、64、128という10進数の値をつけます。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
↑	↑	↑	↑	↑	↑	↑	↑
128	64	32	16	8	4	2	1

そして、1 が置かれたbit番号の値だけを足していって、2 進数を10進数に変換するのです。

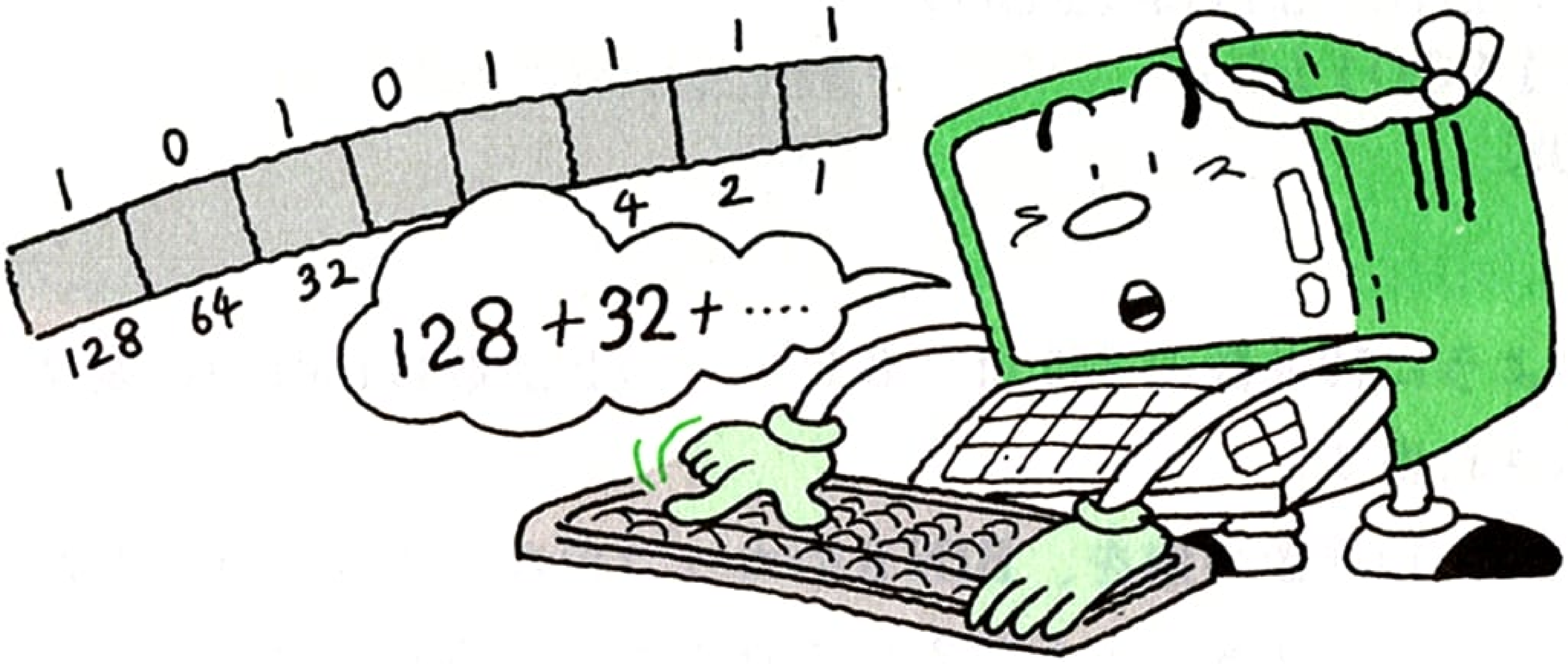
では 8 ビットの 2 進数10101111を、10進数に変換してみることにします。まず、10101111をビット番号のうえに置きます。

1	0	1	0	1	1	1	1
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
↑	↑	↑	↑	↑	↑	↑	↑
128	64	32	16	8	4	2	1

このように 8 ビットの 2 進数を置いたら、1 が置かれているビット番号の値だけを足します。この場合は、bit 0 、bit 1 、bit 2 、bit 3 、bit 5 、bit 7 に 1 が置かれていますから、その値を足すことになります。

$$128+32+8+4+2+1=175$$

したがって、8 ビットの 2 進数10101111は、10進数で 175 となるのです。



10進数を２進数に変換する

MSXベーシックのBIN$を使っての変換

82頁で、８ビットの２進数を10進数に変換しました。ここでは10進数を８ビットの２進数に変換することにします。

10進数を２進数に変換する最も簡単な方法は、ＭＳＸベーシックのBIN＄を使うことです。BIN＄は関数で、つぎのような形をしています。

```
BIN$ (X)
```

BIN＄のカッコのなかのＸに、２進数に変換する10進数を与えると、その10進数が２進数に変換されます。

もちろん、BIN＄は関数ですから、つぎのように、

```
PRINT BIN$ (X)
```

ステートメントにしなければなりません。

まずダイレクトモードで、10進数を２進数に変換してみることにします。２進数に変換する10進数は230とします。したがって、230をＢＩＮ＄（Ｘ）のＸに与えます。

```
print bin$(230)
11100110  ←──２進数に変換されて表示
OK
```

このように10進数の230は、８ビットの２進数11100110に変換されて表示されます。

ただし、10進数の値を、BIN＄のカッコのなかのＸに与えたからといって、かならず８ビットで表示されるとはかぎりません。たとえば

10進数の48は、8ビットの2進数00110000に変換されますが、つぎに示すように、左側の1のまえにあるふたつの0が省略されて、110000というように表示されます。

```
print bin$(48)
110000
OK
```

1のまえの0がふたつ省略されて表示

左側の1のまえに0がひとつあるときは、その0が省略されて表示されます。たとえば10進数の79は、8ビットの2進数01001111に変換されますが、1001111というように表示されます。

```
print bin$(79)
1001111
OK
```

1のまえの0がひとつ省略されて表示

2進数に慣れていれば、このように表示されてもかまわないのですが、2進数に慣れていないととまどってしまいます。このような表示ではなく、きちんと8ビットの2進数で表示させるには、そのように表示させるプログラムを作らなければなりません。

つぎに、そのプログラムを示します。

```
10 INPUT A
20 IF A=0 THEN 50
30 PRINT RIGHT$("00000000"+BIN$(A),8)
40 GOTO 10
50 END
```

プログラムはベーシックなので、詳しいことはベーシックの本を参照してもらうことにして、ここでは簡単に説明しておくことにします。

行番号10のＩＮＰＵＴ文で、変数Ａに２進数に変換する10進数を入力します。

行番号30のカッコのなかの〝００００００００〟＋BIN＄（Ａ）で、変換された２進数に８個の０を足します。そして、ＲＩＧＨＴ＄関数で右から８文字取りだしてＰＲＩＮＴ文で表示させると、かならず８ビットの２進数で表示されます。BIN＄（Ａ），８の８がＲＩＧＨＴ＄関数で取りだす文字数の指定です。

このプログラムは、行番号40のGOTO文で繰り返えしていますから、10進数を２進数に変換することがおわっても、繰り返えされます。そこで、10進数を２進数に変換することがおわったら、０を入力するようにしています。０が入力されると行番号20のＩＦ～ＴＨＥＮ文で行番号50のＥＮＤ文にジャンプしておわりとなります。

つぎに、このプログラムの実行結果を示します。

```
run
? 48
00110000
? 79
01001111
? 24
00011000
```

以上、BIN＄関数を使って、10進数を８ビットの２進数に変換する方法について説明しました。２進数に慣れていないと、変換するときにどうしても間違いがおこりやすいものですが、この方法はＭＳＸパソコンが変換してくれるので、これほど正確な変換はありません。

２進数を10進数に変換するところでも述べましたが、ただこの方法だと、どうして10進数の24が、８ビットの２進数00011000になるのかということがわかりません。10進数の24が８ビットの２進数になるということについては、70頁の２進数の数の表わし方を参照してもらえばわかると思いますが、ここでも説明しておくことにします。

ビット番号の値を選んで1を置く

2進数を10進数に変換する項の84頁で説明したように、8ビットの2進数のそれぞれの桁には、bit 0からbit 7までのビット番号がついています。8ビットの2進数10101011を例にとると、つぎのようになります。

1	0	1	0	1	0	1	1
↓	↓	↓	↓	↓	↓	↓	↓
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

このビット番号には、1、2、4、8、16、32、64、128という10進数の値がつきます。

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
↑	↑	↑	↑	↑	↑	↑	↑
128	64	32	16	8	4	2	1

10進数を8ビットの2進数に変換するには、このビット番号につけた値に基づいて行っていきます。つぎに実際に、10進数の98を8ビットの2進数に変換してみることにします。

ビット番号につけられた値をみたところ、98という値はありません。したがって、98より小さい値で、98に最も近いビット番号の値を選びます。bit 6の64です。そこで、bit 6のところに1を置きます。

	1						
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
128	64	32	16	8	4	2	1

bit 6の64に1を置いたら、98から64を引きます。残りは34となります。そこでつぎに、34の値を選びます。34という値もありませんから、まえとおなじように34よりも小さくて、34に最も近い値を選び、そこに1を置きます。bit 5の32です。

	1	1					
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
128	64	32	16	8	4	2	1

bit 5 の32に 1 を置いたら、34から32を引きます。残りは 2 です。 2 の値はbit 1 ですから、bit 1 に 1 を置きます。

	1	1				1	
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
128	64	32	16	8	4	2	1

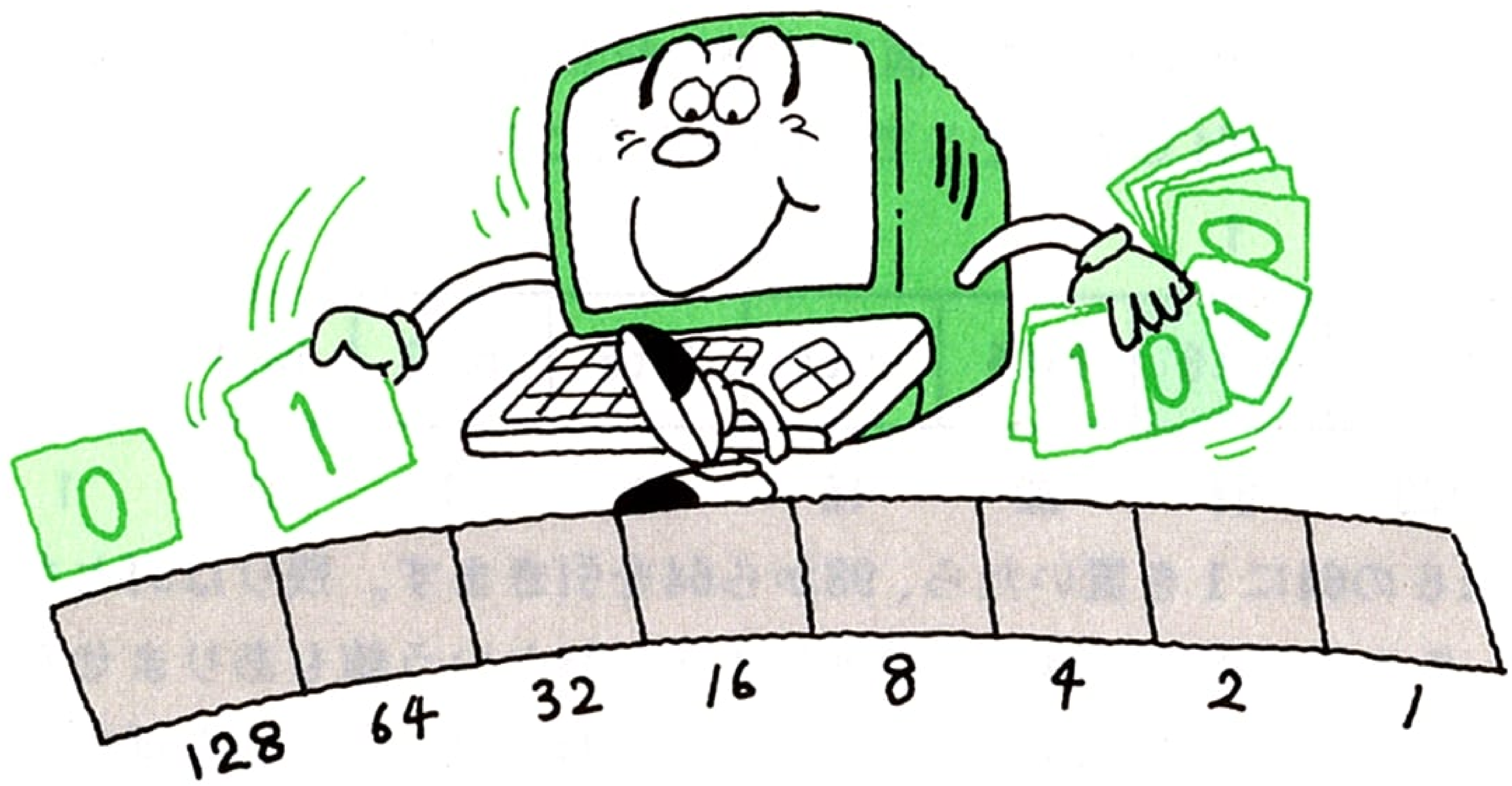
このように、bit 6 、bit 5 、bit 1 に 1 を置くと、 それぞれのビット番号の値は64、32、 2 ですから、

64＋32＋ 2 ＝98

98になります。 1 が置かれないビット番号には、 0 を置きます。

0	1	1	0	0	0	1	0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
128	64	32	16	8	4	2	1

したがって10進数の98は、 8 ビットの 2 進数01100010となるのです。



16進数を10進数に変換する

MSXベーシックの&Hを使って変換

82頁で２進数を10進数に、86頁で10進数を２進数に変換する方法について説明しましたが、マシン語を使うときには、16進数を10進数に変換したり、10進数を16進数に変換する必要が多くなります。そこでここでは、16進数を10進数に変換する方法について説明することにします。10進数を16進数に変換する方法については、94頁を参照してください。

16進数を10進数に変換するときには、97頁に示した16進↔10進変換表をみて行うのがふつうですが、ＭＳＸベーシックには＆Ｈがあります。この＆Ｈを使うと簡単に、しかも正確に16進数を10進数に変換することができます。

＆Ｈは、＆Ｈのあとにつづく０からＦまでの16個の数字で作られた並びが、16進数であることを示します。

＆Ｈ＝０からＦの数字の並びが16進数であることを示す

０からＦまでの16個の数字とは、つぎに示す数字です。このことについては、75頁で説明していますから参照してください。

0　1　2　3　4　5　6　7　8　9　A　B　C　D　E　F

＆Ｈを使って16進数を10進数に変換するには、ダイレクトモードでPRINTのあとに＆Ｈと10進数に変換する16進数を入力すればよいのです。

16進数のＦＦ（10進数で255）までは、97頁に示した10進↔16進変換表をみても簡単に変換できますから、ここでは、16進数のＦＦ以上の数を10進数に変換することにします。まず、16進数３桁の１０Ｆを10進数に変換してみます。

```
print &h10f
271 ←──10進数に変換されて表示
OK
```

ではつぎに、16進数の4桁を10進数に変換してみます。

```
print &h7fff
32767
OK
```

ここまでのところ、16進数はプラスの値で10進数に変換されて表示されました。では、必ずこのようにプラスの値で10進数に変換されて表示されるのかというと、そうではありません。16進数の4桁でも８０００からＦＦＦＦまでは、どれもマイナスの値で表示されます。

```
print &h8000
-32768 ←──マイナスの値で表示
OK
print &ha101
-24319
OK
print &hc000
-16384
OK
```

これは、16進数の８０００からＦＦＦＦまでが、－1から－32768に対応しているからです。

したがって、マイナスの値で表示される16進数の８０００からＦＦＦＦまでの値には、つぎに示すように65536を足します。

```
PRINT &H8000+65536
```

このように65536という値を足して実行させると、マイナスの値ではなく、プラスの値で10進数に変換されて表示されます。

つぎに実行結果を示します。

```
print &h8000+65536
32768 ←――――― 65536を足すとプラスの値で表示
OK
print &ha101+65536
41217
OK
```

いちいちＰＲＩＮＴと＆Ｈ、それに65536を入力していたのでは大変なので、つぎに16進数を10進数に変換するプログラムを示します。

```
10 INPUT A$
20 IF A$="0" THEN 70
30 A=VAL("&h"+A$)
40 IF A<0 THEN A=A+65536!
50 PRINT A
60 GOTO 10
70 END
```

では、簡単にプログラムの説明をします。行番号10のＩＮＰＵＴ文で、文字変数Ａ＄に10進数に変換する16進数を入力します。入力された16進数は、行番号30のＶＡＬ関数のカッコのなかの〝＆ｈ〟＋Ａ＄で、Ａ＄の内容と＆Ｈが連結されます。こうすることで、ＶＡＬ関数で数値に変換するとき、16進数の数値として変換されます。ＶＡＬ関数で変換された数値は、変数Ａに代入されます。

16進数４桁の８０００以降はマイナスになりますから、行番号40のＩＦ文で判断させて、THEN文のＡ＋65536で、マイナスの値をプラスの値にして、行番号50のＰＲＩＮＴ文で画面に表示します。

なお、16進数を２進数に変換することがおわったら、０を入力します。すると行番号20のＩＦ～ＴＨＥＮ文で、行番号70のＥＮＤ文にジャンプしておわります。

10進数を16進数に変換する

MSXベーシックのHEX$を使って変換する

91頁で、16進数を10進数に変換する方法について説明しましたから、ここでは、10進数を16進数に変換する方法について説明します。

16進数を10進数に変換する項でも述べましたが、10進数を16進数に変換するときも、97頁に示した10進↔16進変換表をみて行うのがふつうです。ただ10進↔16進変換表は、16進数の００からＦＦまでは一欄表になっていますから、それをみても簡単に変換することができますが、数が大きくなった場合、慣れていないと簡単に変換することができません。ＭＳＸベーシックにはHEX＄がありますから、このＨＥＸ＄を使うと、簡単に10進数を16進数に変換することができます。

HEX＄は関数で、つぎのような形をしています。

HEX$ (X)

HEX＄のカッコのなかのＸに10進数を与えると、その10進数が16進数に変換されます。もちろん、HEX＄は関数ですから、

PRINT HEX$ (X)

というようにステートメントにしなければなりません。

まずダイレクトモードで、10進数を16進数に変換してみることにします。255までは10進↔16進変換表をみればすぐわかりますから、ここでは255をこえる数にします。最初は10進数の512です。したがって、512をHEX＄のカッコのなかのＸに与えます。

```
print hex$(512)
200 ←――――――16進数に変換されて表示
Ok
```

つぎは、10進数の48672を16進数に変換します。

```
print hex$(48672)
BE20
OK
```

このような大きな数の10進数を16進数に変換するときには、HEX$を使うと簡単に、しかも間違いなく16進数に変換することができます。もちろん、小さい数の10進数を16進数に変換するときに使っても便利であることはいうまでもありません。

ただし、HEX$で16進数に変換することができるのは、10進数の65535までで、65536を変換させようとすると、Overflowになります。

なおダイレクトモードだと、10進数をいくつも16進数に変換するときに不便です。そこで、10進数を16進数に変換するプログラムを示します。

```
10 INPUT A
20 IF A=0 THEN 50
30 PRINT HEX$(A)
40 GOTO 10
50 END
```

では、簡単にプログラムの説明をしておきます。

行番号10のINPUT文で変数Aに入力された10進数は、行番号の30のPRINT文のHEX$関数のカッコのなかのAに送られて、16進数に変換されて表示されます。

このプログラムでは、行番号40のGOTO文で繰り返えしていますから、10進数を16進数に変換することがおわったら、0を入力します。0を入力すると、行番号20のIF～THEN文で判断されて、行番号50のEND文にジャンプしておわりとなります。

10進↔16進変換表を使って変換する

16進数を10進数に、10進数を16進数に

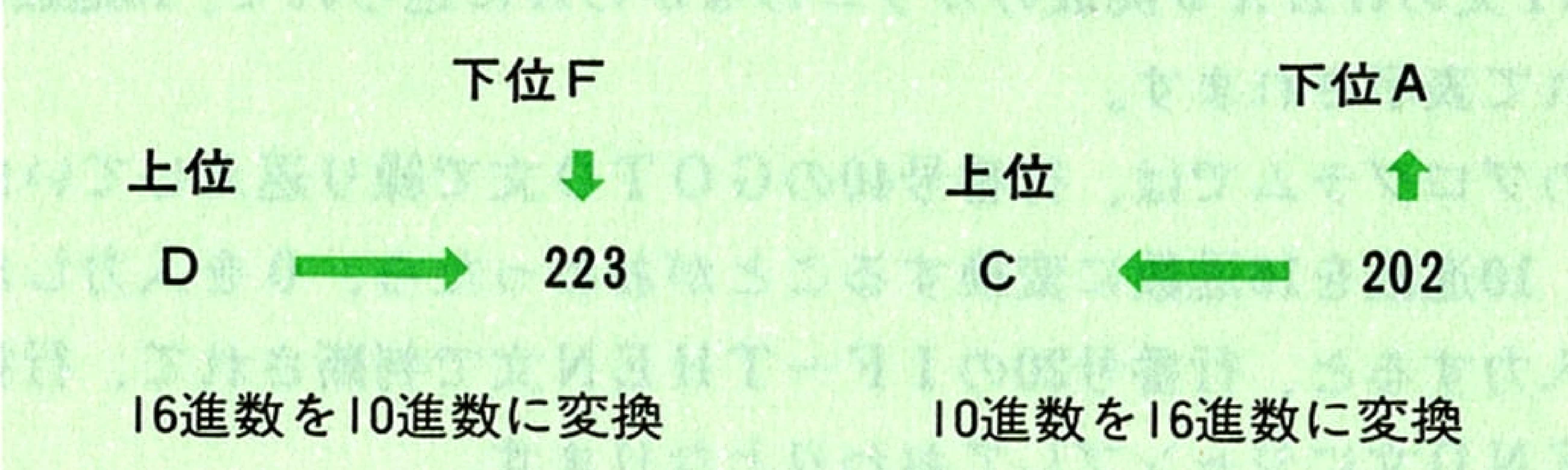
91頁で16進数を10進数に、94頁で10進数を16進数に変換しましたが、ベーシックの＆ＨやＨＥＸ＄を使わないで変換するには、つぎの頁に示した10進↔16進変換表を使って変換します。10進↔16進変換表の使い方は簡単です。

まず、16進数を10進数に変換してみます。表の左横に示されている０からＦまでは16進数の上位桁、表の上に示されている０からＦまでは16進数の下位の桁です。16進数のＤＦを例にとると、Ｄが上位の桁、Ｆが下位の桁ということになります。

ＤＦ←――――Ｄが上位の桁、Ｆが下位の桁

したがって、この16進数ＤＦを10進数に変換するには、表の左横に示されている上位の桁Ｄの位置から、横にたどっていきます。一方、表の上に示されている下位の桁Ｆの位置から、下にたどっていきます。そして、交わったところの値が、16進数ＤＦの10進数となります。この場合は223ですから、16進数ＤＦは10進数で223です。

10進数を16進数に変換するには、16進数を10進数に変換したときと逆のことを行えばよいのです。たとえば､10進数の202を16進数に変換するには､202の位置から、上位の桁が示されている表の横をみます。



Ｃです。一方は202の位置から、下位の桁が示されている表の上をみます。Ａです。したがって10進数の202は、16進数でＣＡということになります。

２番目に示した表は、ＦＦＨ(10進数で255）以上の値を変換するときに使います。ＦＦＨ以上の10進↔16進変換は、91頁と94頁で説明した方法を使ったほうが便利なので、ここでは説明を省略します。

10進↔16進変換表

上位桁＼下位桁	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

桁＼最上位×	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
×00	256	512	768	1024	1280	1536	1792	2048	2304	2560	2816	3072	3328	3584	3840
×000	4096	8192	12288	16384	20480	24576	28672	32768	36864	40960	45056	49152	53248	57344	61440
×0000	65536														

2進数を16進数に変換する

MSXベーシックの&BとHEX$を使う

スプライト面にスプライトを表示するとき、スプライトパターンデータを定義します。ベーシックの場合は、スプライトパターンデータは1と0の2進数で定義しましたが、マシン語の場合は16進数で定義しなければなりません。16進数で定義するときは、一度1と0でスプライトパターンを書いてから、16進数になおします。したがって、2進数を16進数に変換する必要があるので、説明しておきます。

16進数を2進数に変換する方法については、102頁を参照してください。

2進数を16進数に変換する最も簡単な方法は、82頁と94頁で説明したMSXベーシックの&Bと、HEX$のふたつを使うことです。

&B＝1と0の並びが2進数であることを示す
HEX$（X）＝10進数を16進数に変換する

&BとHEX$のふたつを使って2進数を16進数に変換するには、つぎのようにHEX$のカッコのなかのXに、&Bと16進数に変換する2進数を与えればよいのです。

```
print hex$(&b11111111)   ←カッコのなかに&Bと変換する2進数を与える
FF   ←16進数に変換されて表示
OK
print hex$(&b10011110)
9E
OK
```

これが、MSXベーシックの&BとHEX$を使って、2進数を16

進数に変換する方法ですが、このようにダイレクトモードで、いちいち入力するのはめんどうですから、これをプログラムにすることにします。つぎに示すのが、そのプログラムです。

```
10 INPUT A$
20 IF A$="0" THEN 60
30 B$=HEX$(VAL("&b"+A$))
40 PRINT B$;"H"
50 GOTO 10
60 END
```

このプログラムはＭＳＸベーシックなので、簡単に説明します。

行番号10のＩＮＰＵＴ文で、文字変数Ａ＄に16進数に変換する２進数を入力します。文字変数Ａ＄に入力された２進数は、行番号30のＰＲＩＮＴ文のＶＡＬ（〝＆ｂ〟＋Ａ＄）のＡ＄に送られて＆Ｂと連結されます。そして、ＶＡＬ関数で２進数の数値として10進数に変換されます。ＶＡＬ関数で10進数に変換されると、その数値は、つぎにＨＥＸ＄関数で16進数に変換されて、文字変数Ｂ＄に代入されます。

Ｂ＄に代入された16進数は、行番号40のＰＲＩＮＴ文でＨと一緒に表示されます。

このプログラムは、行番号50のＧＯＴＯ文で繰り返えしていますから、２進数の16進数への変換がおわったら、０を入力します。０を入力すると、行番号20のＩＦ～ＴＨＥＮ文で判断されて、行番号60のＥＮＤ文にジャンプしておわります。つぎに実行結果を示します。

```
? 1010
AH
? 10011110
9EH
? 01001010
4AH
```

以上、ＭＳＸベーシックの＆ＢとＨＥＸ＄を使って、２進数を16進数に変換しましたが、つぎに２進数をふたつにわけて、16進数に変換する方法を説明することにします。

2進数を上位ビット、下位ビットのふたつに

84頁で８ビットの２進数には、bit１からbit７までのビット番号がついていると説明しました。

1	0	0	1	1	1	0	0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

最上位ビット　　　　　　　　　　　　　　　　最下位ビット

８ビットの２進数を16進数に変換するには、このbit０からbit７までを、つぎのようにふたつに分けます。

1	0	0	1	1	1	0	0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
上位ビット				下位ビット			

このようにふたつに分けた場合、bit０からbit３までを下位ビット、bit４からbit７までを上位ビットといいます。

ビット番号をこのように分けると、さきにビット番号のうえに置いた８ビットの２進数10011100は、うえに示したように、

1001	1100
上位ビット	下位ビット

４ビットずつになりますから、上位ビットの１００１と下位ビットの１１００にあたる16進数を、80頁に示した４ビットの２進↔16進対応表をみて探します。２進↔16進対応表をみると、

２進数		16進数
1001	➡	9
1100	➡	C

となっていますから、上位ビットの１００１と下位ビットの１１０

０に、16進数の９とＣを、つぎのようにあてはめます。

```
  1001          1100
   ↓             ↓
   9             C  ←──16進数をあてはめる
    ↘    9CH    ↙   ←──ふたつをまとめる
```

そして、９とＣをまとめて９ＣＨとします。これで８ビットの２進数10011100は、16進数で９ＣＨということになります。

もうひとつ、８ビットの２進数11110001を16進数に変換してみることにします。11110001をふたつに分けると、

```
  1111        0001
 上位ビット    下位ビット
```

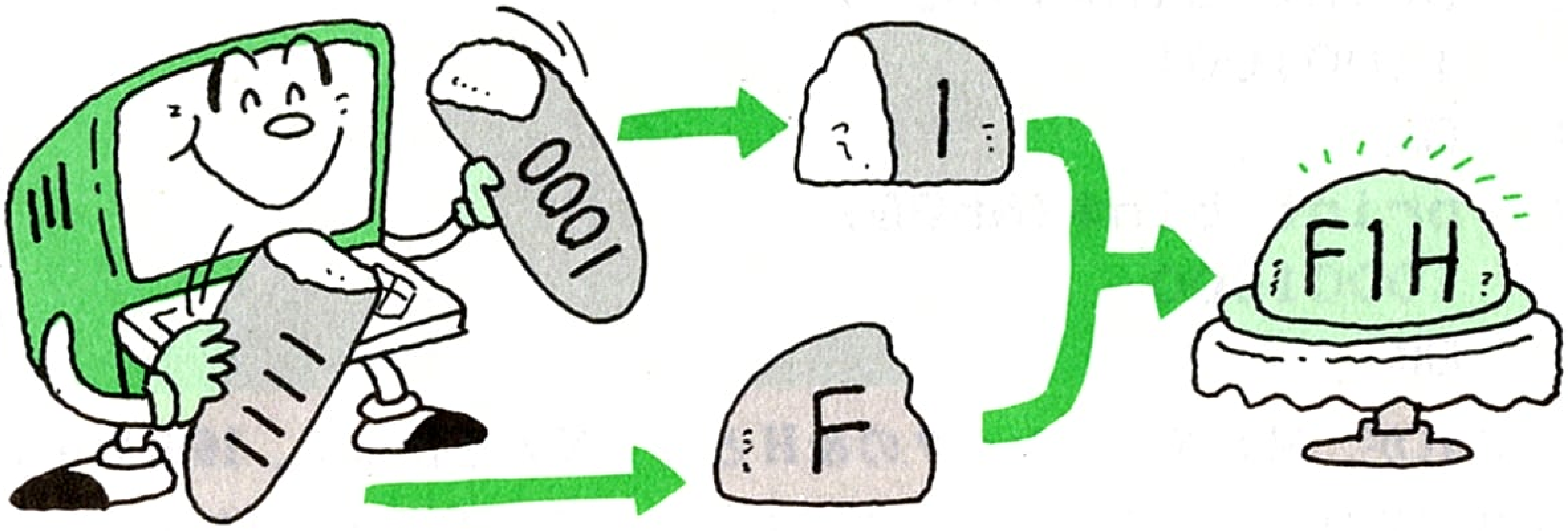
となりますから、80頁の４ビットの２進↔16進対応表をみます。

２進数		16進数
1111	➡	F
0001	➡	1

となっていますから、この16進数のＦと１を、上位ビットの１１１１と下位ビットの０００１にあてはめます。

```
  1111          0001
   ↓             ↓
   F             1  ←──16進数をあてはめる
    ↘    F1H    ↙   ←──ふたつをまとめる
```

これで８ビットの２進数11110001は、16進数でＦ１Ｈとなります。



16進数を２進数に変換する

MSXベーシックの&HとBIN$を使う

98頁で、２進数を16進数に変換する方法について説明しました。ここではその逆に、16進数を２進数に変換する方法について説明します。

16進数を２進数に変換する最も簡単な方法は、86頁と91頁で説明したＭＳＸベーシックの＆ＨとＢＩＮ＄を使うことです。

＆Ｈ＝０からＦの数字の並びが16進数であることを示す
ＢＩＮ＄（Ｘ）＝10進数を２進数に変換する

なお、０からＦの数字の並びについては、75頁の「16進数とは」の項を参照してください。

＆ＨとＢＩＮ＄のふたつを使って16進数を２進数に変換するには、ＢＩＮ＄のカッコのなかのＸに、＆Ｈと２進数に変換する16進数２桁を与えればよいのです。

```
print bin$(&h0a)   ←カッコのなかに&Hと16進数を与える
1010               ←2進数に変換されて表示
OK
print bin$(&hc9)
11001001
OK
print bin$(&h88)
10001000
OK
```

これが、ＭＳＸベーシックの＆ＨとＢＩＮ＄を使って、16進数を２進数に変換する方法です。ダイレクトモードでは、いちいちステート

メントを入力しなければならないので、つぎのようにプログラムにするとよいでしょう。

```
10 INPUT A$
20 IF A$="0" THEN 60
30 B$=BIN$(VAL("&h"+A$))
40 PRINT RIGHT$("00000000"+B$,8)
50 GOTO 10
60 END
```

このプログラムはベーシックなので説明の必要はないと思いますが、簡単に説明しておくことにします。

行番号10のINPUT文で、文字変数A＄に2進数に変換する16進数を入力します。入力する16進数は、16進数2桁です。なぜなら、MSXパソコンは、8ビットの2進数をひとつのまとまりとして扱うからで、8ビットは16進数2桁だからです。

文字変数A＄に代入された16進数は、行番号30のVAL（〝&h〃+A＄）のA＄に送られて&Hと連結されます。そして、VAL関数で16進数の数値として10進数に変換されます。VAL関数で10進数に変換されると、その数値は、つぎにBIN＄関数で2進数に変換されて、文字変数B＄に代入されます。

B＄に代入された2進数は、行番号40のPRINT文のRIGHT＄(〝00000000〃+B＄,8)のB＄に送られて、8個の0と連結されます。そして、RIGHT＄関数で右から8個取りだして表示します。

このように8個の0と連結して、RIGHT＄関数で8個取りだすのは、87頁で説明しているように、左側の1のまえにつく0を省略させないで表示するためです。

このプログラムは、行番号50のGOTO文で繰り返えしていますから、16進数を2進数に変換がおわったら、0を入力します。0を入力すると、行番号20のIF～THEN文で判断されて、行番号60のEND文にジャンプしておわります。つぎに実行結果を示します。

```
? 3e        ←16進数2桁を入力
00111110    ←2進数に変換されて表示
OK
? 85
10000101
OK
? 32
00110010
OK
```

以上が、ＭＳＸベーシックの＆ＨとＢＩＮ＄を使って、16進数を２進数に変換する方法です。つぎに16進数を１桁ずつに分けて、２進数に変換する方法を説明することにします。

16進数を上位の桁と下位の桁に分ける

８ビットの２進数は、16進数２桁です。

３ＥＨ　ＥＡＨ　３２Ｈ　Ｅ４Ｈ　Ｆ６Ｈ

16進数２桁を８ビットの２進数に変換するには、つぎのように16進数を１桁ずつに分けます。16進数の３ＥＨを例にとると、つぎのようになります。



左側の１桁を上位の桁、右側の１桁を下位の桁といいます。このように上位の桁と下位の桁の１桁ずつに分けたら、80頁に示した４ビットの２進↔16進対応表をみます。２進↔16進対応表をみると、

２進数		16進数
００１１	←	３
１１１０	←	Ｅ

となっていますから、００１１を上位の桁の３に、１１１０を下位

の桁のＥにあてはめます。

```
 3             E
 ↓             ↓
0011          1110 ←4ビットの2進数を上位、
                     下位の桁にあてはめる
    ↘       ↙
    00111110 ←ひとつにまとめる
```

このようにあてはめたら、００１１と１１１０をひとつにまとめます。これで16進数の３ＥＨは、８ビットの２進数00111110ということになります。

では、もうひとつ16進数のＦ６Ｈを、８ビットの２進数に変換してみます。16進数のＦ６Ｈを、上位の桁と下位の桁に分けます。

```
F        6 ←16進数2桁を1桁ずつに分ける
```

そして、80頁の２進↔16進対応表をみます。

2進数		16進数
1111	←	F
0110	←	6

したがって、上位の桁のＦに１１１１、下位の桁の６に０１１０をあてはめて、つぎに、それをひとつにまとめます。

```
 F             6
 ↓             ↓
1111          0110 ←上位の桁と下位の桁に
                     あてはめる
    ↘       ↙
    11110110 ←ひとつにまとめる
```

これで16進数のＦ６Ｈは、８ビットの２進数で11110110になります。



マシン語でプログラムを書くには

マシン語は、16進数2桁で書く

ＭＳＸパソコンの内部では、１と０の８個の並びが行き交っていますが、私たちにとってのマシン語は、この１と０の並びである２進数で書き表わすのではなくて、16進数２桁で書き表わします。

```
3E 82
06 36
90
32 10 E0
C9
```

このマシン語のプログラムはわずか９バイトですが、それでも立派なプログラムです。どんなことをさせるプログラムかというと、

130−54＝

という引き算のプログラムです。マシン語のプログラムの書き方を長いマシン語のプログラムを使って説明するとややこしくなるので、最も短く、簡単なプログラムを使うことにしました。

ところで、このマシン語のプログラムが９バイトでできているというのは、16進数２桁が１バイト（８ビット）で、９個あるからです。

ではマシン語のプログラムは、なぜ16進数２桁ずつで書き表わされるのでしょうか。なぜならＭＳＸパソコンは、１と０の８個の並び、つまり８ビット（１バイト）の２進数をひとつのまとまりとして、メモリに記憶したり、やりとりしたりしています。８ビット（１バイト）の２進数は16進数２桁に変換されますから、16進数では、２桁が８ビット（１バイト）になります。したがって、マシン語のプログラムは、

16進数2桁ずつで書き表わされるのです。

16進数2桁＝8ビット（1バイト）

最初はアセンブリ言語で書く

さて、まえの頁に示したマシン語のプログラムを書くには、直接、16進数でプログラムを書いていくわけではありません。まえの頁のわずか9バイトのマシン語のプログラムをみてもわかるように、直接、16進数という数字だけを並べてプログラムを書くことは、とてもできません。そこで、17頁で説明しているアセンブリ言語でプログラムを書いてから、それをマシン語に書きなおしていくという方法をとります。

では、まえの頁のマシン語のプログラムを、アセンブリ言語で書いて、それをどのようにしてマシン語に書きなおすのか説明することにします。まえの頁に示したマシン語のプログラムは、引き算のプログラムです。引き算を行わせるには、まず引かれる数をAレジスタに入れます。数値をレジスタやメモリに入れる命令は、ＬＤ命令です。LDは、英語のload（ロード）の略記号です。英語のloadは「入れる」という意味です。

LD命令＝レジスタやメモリに数値を入れる命令



Aレジスタや B レジスタなどのレジスタは、レジスタの名前であるAとかBで書き表わします。

レジスタ＝レジスタの名前であるAとかBで書き表わす

引かれる数の130は10進数ですから、これを16進数になおします。

130＝82H←引かれる数を16進数になおす

アセンブリ言語では、数値にはHをつけて書き表わします。

Aレジスタに82Hを入れる命令の書き方は、LD命令が最初で、そのあとにAレジスタ、そしてコンマで区切って82Hを書きます。

LD　A，82H←Aレジスタに82Hを入れよ

これでLD命令で、Aレジスタに82Hが入れられて記憶されます。

引く数は、Aレジスタ以外のレジスタに入れます。引かれる数はAレジスタに入れると決まっていますが、引く数は、Aレジスタ以外のレジスタなら、どのレジスタでもかまいません。ここでは、引く数はBレジスタに入れることにします。Bレジスタに入れる命令は、LD命令です。引く数の54は10進数なので、16進数になおします。なお、レジスタについては144頁を参照してください。

54＝36H←引く数を16進数になおす

書き方は、Aレジスタに82Hを入れたときとおなじです。AがBに、82Hが36Hに変わるだけです。

LD　B，36H←Bレジスタに36Hを入れよ

このように16進数2桁の数値を、LD命令でレジスタに入れることを、8ビット・ロード命令といいます。

16進数2桁の数値をレジスタに入れる＝8ビット・ロード命令

Aレジスタに82H、Bレジスタに36Hを入れたら、つぎはAレジスタの内容82Hから、Bレジスタの内容36Hを引きます。

引く命令は、SUB命令です。SUBは、英語のsubtract（サブトラクト）の略記号です。英語のsubtractは、「引く」という意味です。

SUB命令＝レジスタの内容から引く命令

SUB命令で引き算を行うときは、引く数を入れたレジスタの名前

だけを、ＳＵＢ命令のあとに書きます。引かれる数を入れたＡレジスタは書きません。したがってこの場合は、Ｂレジスタに引く数を入れましたから、ＳＵＢ命令のあとにはＢとだけ書きます。

ＳＵＢ　Ｂ←――引く数を入れたレジスタの名前だけを書く

ＳＵＢ命令では、Ａレジスタの内容から引くと決まっているので、Ａレジスタは書かないのです。したがって、つぎのようには書かないので注意してください。

ＳＵＢ　Ａ，Ｂ←――このようには書かない

これでＳＵＢ命令で、Ａレジスタの内容８２Ｈから、Ｂレジスタの内容３６Ｈが引かれます。

８２Ｈ－３６Ｈ＝４ＣＨ（10進数で76）

引き算の結果の４ＣＨは、Ａレジスタに記憶されて残ります。Ａレジスタに記憶されていたまえの内容８２Ｈは、４ＣＨで消されてしまいます。レジスタもベーシックの変数とおなじで、新しく値が入れられると、まえに入れられていた値は消されてしまって、新しく入れられた値が記憶されるのです。

引き算の結果がどうなったかをみるために、Ａレジスタの内容４ＣＨを、Ｅ０１０番地のメモリに入れます。ここでは説明の都合上、引き算の結果の答え４ＣＨとあきらかにしていますが、実際には引き算の結果はわからないわけです。したがって、Ａレジスタに記憶されている内容を、メモリ番地を指定して、そのメモリに入れて記憶させ、あとからＤコマンドでメモリ番地を指定してみるわけです。

この場合は、Ｅ０００番地のメモリから、マシン語のプログラムを入力して実行させるので、Ａレジスタの内容をＥ０１０番地のメモリに入れて記憶させることにします。したがって、メモリ番地はＥ０１０と指定します。メモリに記憶させる場合も、ＬＤ命令です。

メモリにＡレジスタの内容を記憶させる書き方は、最初にＬＤ命令、つぎに記憶させるメモリ番地、最後にコマンドで区切ってＡレジスタの順です。ただし、つぎのようには書きません。

ＬＤ　Ｅ０１０Ｈ，Ａ←――このように書かない

つぎのように、E010Hをカッコでかこみます。

```
LD  (E010H), A
```
←メモリ番地をカッコでかこむ

これでAレジスタの内容が、指定したメモリ番地に記憶されます。

プログラムの最後は、マシン語モニタに戻すRET命令を書きます。RETは、英語のRETURN（リターン）の略記号です。英語のRETURNは、「戻る」という意味です。

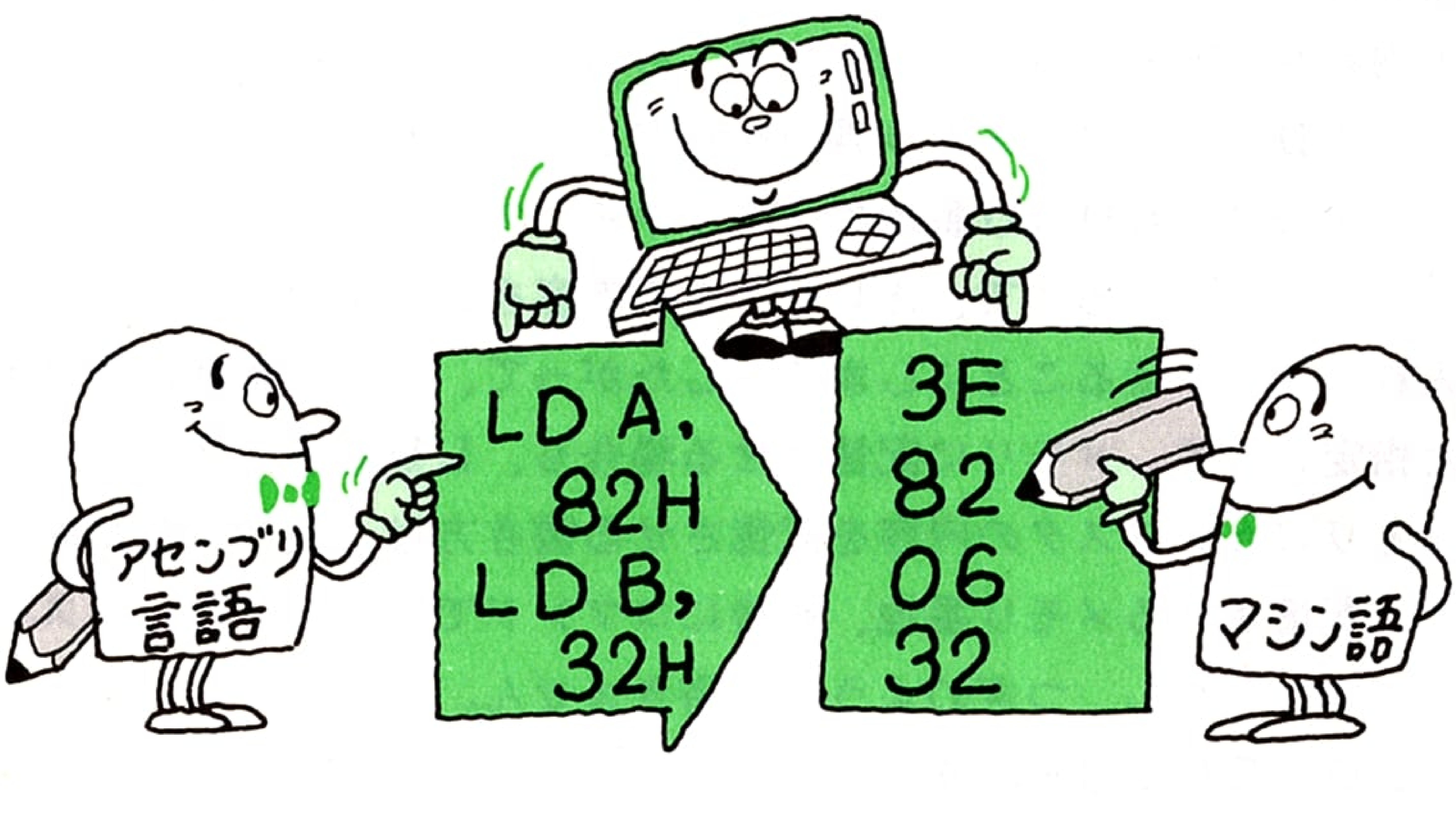
RET命令＝サブルーチンから戻す命令

MSXパソコンの場合は、マシン語モニタはベーシックで作られていて、マシン語のプログラムをサブルーチンとして処理しています。したがって、MSXパソコンの場合は、マシン語のプログラムの最後にはかならず、RET命令を書かなければならないのです。

アセンブリ言語をマシン語に書きなおす

アセンブリ言語でプログラムを書きおわったら、それをマシン語に書きなおします。手作業でマシン語に書きなおしていくことを、**ハンドアセンブル**、またハンドを省略して、**アセンブル**ともいいます。

ハンドアセンブル＝手作業でマシン語に書きなおす



アセンブリ言語をマシン語に変換するプログラムがあります。このプログラムを使ってMSXパソコンで、アセンブリ言語をマシン語になおすことを、**アセンブラ**といいます。

アセンブラ＝プログラムでアセンブリ言語をマシン語になおす

ここでは、ハンドアセンブルです。ハンドアセンブルは、142頁に示した「Z－80命令表」を使って行います。下の表に、ここで使うものだけを「Z－80命令表」のなかから抜きだして示します。

さて、さきに書いたアセンブリ言語のプログラムをマシン語になおしていくので、つぎにまとめて示します。

```
LD  A, 82H
LD  B, 36H
SUB  B
LD  (E010H), A
RET
```

まず、1行目のアセンブルです。LD　A，82Hというアセンブリ言語をマシン語になおすには、表の横に示されているLD　A，×と書かれたところをみます。下の表では1行目の欄です。

LD　A，×

この場合、LD　A，×の×のところは、アセンブリ言語ではつぎ

	A	B	C	D	E	H	L	n
LD　A，×	7F	78	79	7A	7B	7C	7D	3E n
LD　B，×	47	40	41	42	43	44	45	06 n
SUB　×	97	90	91	92	93	94	95	D6 n
LD(nn)，×	32 n n							
RET	C9							

のように８２Ｈとなっていますから、

LD　A, 82H

⬇

LD　A, ×

まえの頁の表のうえの欄に示されたｎの項をみます。ｎは１バイトの数値を示します。１バイトは、16進数２桁です。

このｎの項と左の欄のＬＤ　Ａ，×と交わったところが、ＬＤ　Ａ，８２Ｈのマシン語となります。

3E

n

ｎには、８２Ｈをそのままあてはめます。したがって、１行目は、

LD　A, 82H

⬇

3E　82

とアセンブルします。

２行目のＬＤ　Ｂ，３６Ｈというアセンブリ言語をマシン語になおすには、ＬＤ　Ｂ，×の欄をみます。

LD　B, ×

この場合もＬＤ　Ｂ，×の×のところには、３６Ｈという１バイトの数値がきますから、ＬＤ　Ｂ，×の欄とｎの項とが交わったところが、ＬＤ　Ｂ，３６Ｈのマシン語となります。

06

n

ｎには、３６Ｈをそのままあてはめます。したがって、２行目は、

LD　B, 32H

⬇

06　36

とアセンブルします。

３行目のＳＵＢ　Ｂというアセンブリ言語をマシン語になおすには、ＳＵＢ　×の欄をみます。

SUB　×

この場合は、ＳＵＢ　×の×のところはレジスタのＢです。

SUB　B

⬇

SUB　×

したがって、表のうえに示されたＢの項をみます。表のうえに示されたＡからＬまでは、レジスタの名前を示しています。このＢの項とＳＵＢ　×の欄が交わったところが、ＳＵＢ　Ｂのマシン語ですから、ＳＵＢ　Ｂは９０とアセンブルします。

SUB　B

⬇

90

4行目のＬＤ（Ｅ０１０Ｈ），Ａというアセンブリ言語をマシン語になおすには、ＬＤ（nn），×の欄をみます。

LD（nn），×

この場合、ＬＤ（nn），×の×のところは、レジスタのＡです。したがって、ＬＤ（nn），×の欄とＡの項が交わったところが、ＬＤ（Ｅ０１０Ｈ），Ａのマシン語です。

32

nn

nnは2バイトの数値を示しています。2バイトの数値は、16進数4桁です。メモリ番地はＥ０１０Ｈですから、16進数の4桁です。

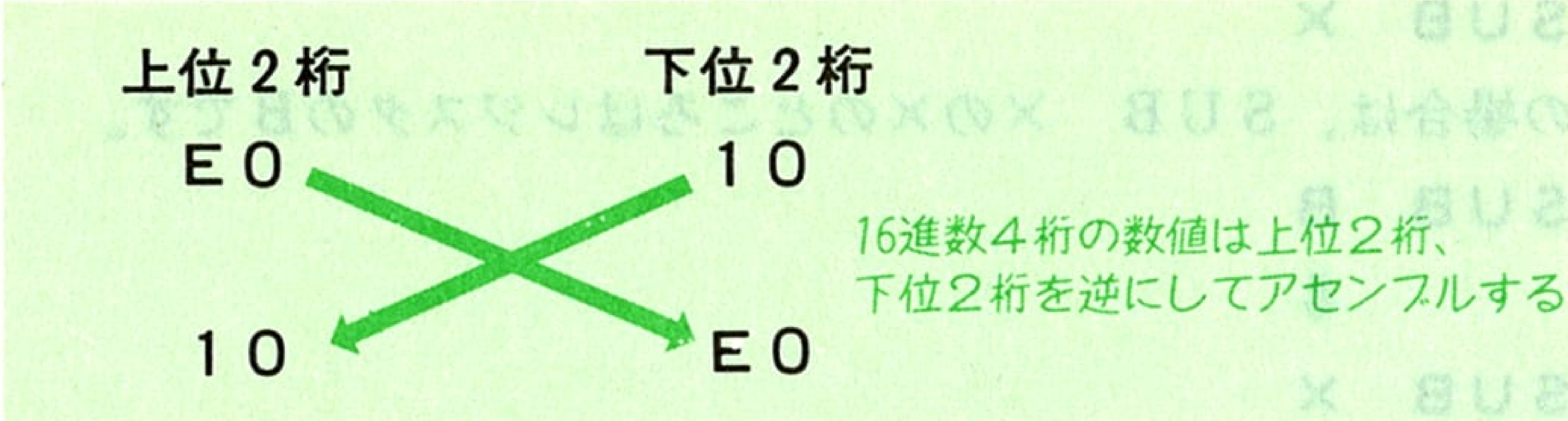
したがって、nnにはＥ０１０Ｈをそのままあてはめますから、

LD　(E010H)，A

⬇

32　10　E0

とアセンブルします。このときに注意しなければならないことは、Ｅ０１０Ｈというような16進数4桁の数値は、つぎに示すように、上位2桁をあとに、下位2桁をさきにするということです。このことを間違えると暴走することにもなります。



最後はＲＥＴです。戻るという場合もいろいろありますが、ふつうサブルーチンから戻るというときのＲＥＴのマシン語はＣ９ですから、

RET
↓ ← ふつうサブルーチンから戻るというときのマシン語はC9
C9

とアセンブルします。これでアセンブルはおわりです。

つぎにアセンブルしたマシン語のプログラムを示します。

マ シ ン 語	アセンブリ言語
３Ｅ　８２	ＬＤ　Ａ，８２Ｈ
０６　３６	ＬＤ　Ｂ，３６Ｈ
９０	ＳＵＢ　Ｂ
３２　１０　Ｅ０	ＬＤ　（Ｅ０１０Ｈ），Ａ
Ｃ９	ＲＥＴ

マシン語をメモリ上に割りつける

アセンブリ言語をマシン語にアセンブルしたら、それでマシン語のプログラムができあがったのかというと、そうではありません。アセンブルがおわったら、マシン語のプログラムをメモリ上に割りつけなければなりません。つまり、何番目のメモリから入力するかを決めて、マシン語にメモリ番地をつけるわけです。

Ｅ０００番地のメモリからマシン語のプログラムを割りつけるときは、Ｅ０００番地からマシン語につけていきます。ひとつのマシン語に対してメモリ番地はひとつですから、Ｅ０００番地は３Ｅ、Ｅ００１番地は８２、Ｅ００２番地は０６というようになります。



なぜ、ひとつのマシン語にひとつのメモリ番地なのかというと、ひとつのメモリに記憶できるのは8ビット（1バイト）だからです。16進数2桁で8ビット（1バイト）です。

ところでメモリ番地は、マシン語のひとつひとつにつけると説明しましたが、ふつうマシン語のプログラムは命令ごとに並んでいます。したがってメモリ番地はつぎのように、最初のマシン語のメモリ番地だけが書かれることになりますから、マシン語のプログラムでは、メモリ番地はとびとびになっています。

メモリ番地	マ シ ン 語
E000	3E 82
E002	06 36
E004	90
E005	32 10 E0
E008	C9

以上で、マシン語のプログラムの書き方の説明はおわりです。

このプログラムをGコマンドで実行させたあと、Dコマンドと、E010とE017を入力すると、E010番地のメモリに記憶させた、引き算の結果の4CHが表示されます。

2バイトの数値は上位2桁と下位2桁を逆にする

上位2桁と下位2桁を逆にする

マシン語ではメモリ番地の指定など2バイトの数値をよく使います。

```
CALL  008DH ←――― 008D番地のルーチンを呼びだす
LD  (E010H), A ← Aレジスタの内容をE010番地のメモリに入れる
JP  EC01H ←――― EC01番地のメモリにジャンプ
LD  A, (E070H) ← E070番地のメモリの内容をAレジスタに入れる
```

2バイトの数値とは、うえに示した００８ＤＨとかＥ０１０Ｈとかいう16進数4桁の数値です。

この2バイトの数値をマシン語にアセンブルするとき、ＭＳＸパソコンでは、上位2桁と下位2桁を逆にすることに決まっています。

2バイトの数値＝上位2桁、下位2桁を逆にする

16進数4桁の上位2桁、下位2桁とは、００８ＤＨを例にとると、

```
 00          8D
 ↑           ↑
上位2桁     下位2桁
```

左側の００が上位2桁、右側の８Ｄが下位2桁となります。これをマシン語にアセンブルするときは、上位2桁があと、下位2桁をさきにしなければなりません。したがって、つぎのようになります。

```
00      8D
   ╲  ╱
    ╳        上位2桁をあと、下位2桁をさきにする
   ╱  ╲
8D      00
```

このことは大切なことなので、おぼえておかなければなりません。

マシン語のプログラムをカセットテープに記録

カセットテープに記録するには、BSAVE

ベーシックのプログラムができあがると、カセットテープに記憶して保管します。マシン語のプログラムの場合もおなじように、カセットテープに記録して保管しなければなりません。マシン語は、リセットボタンを押してＭＳＸパソコンを初期状態に戻しても、メモリから消えませんが、電源を切るとメモリから消えてしまうからです。

ベーシックのプログラムを、カセットテープに記録するコマンドはCSAVEです。マシン語のプログラムを、カセットテープに記録するコマンドはＢＳＡＶＥです。

BSAVE＝マシン語をカセットテープに記録

BSAVEは、つぎのような形をしています。

ＢＳＡＶＥ〝〈ファイルスペック〉〟，〈開始アドレス〉，〈終了アドレス〉

ファイルスペックには、ＣＡＳとファイル名を指定します。たとえばファイル名を「ufo」とするときは、つぎのように指定します。

〝ｃａｓ：ｕｆｏ〟

〈開始アドレス〉には、マシン語のプログラムが記憶されている最初のメモリ番地を、〈終了アドレス〉には、最後のメモリ番地を指定します。たとえば、マシン語のプログラムが、Ｅ０００番地のメモリからＥ０５０番地のメモリに記憶されていたとすると、

Ｅ０００ ← 開始アドレス　　Ｅ０５０ ← 終了アドレス

となります。ただし、E000、E050のまえには、つぎのように16進数であることを示す&Hをつけなければなりません。

BSAVE "CAS:ufo", &HE000, &HE050

では、つぎのマシン語のプログラムで、実際にカセットテープに記録してみることにします。このマシン語のプログラムを実行させるとE050番地からE05B番地までのデータで作られるキャラクタが、スプライト面を左から右へ移動していきます。

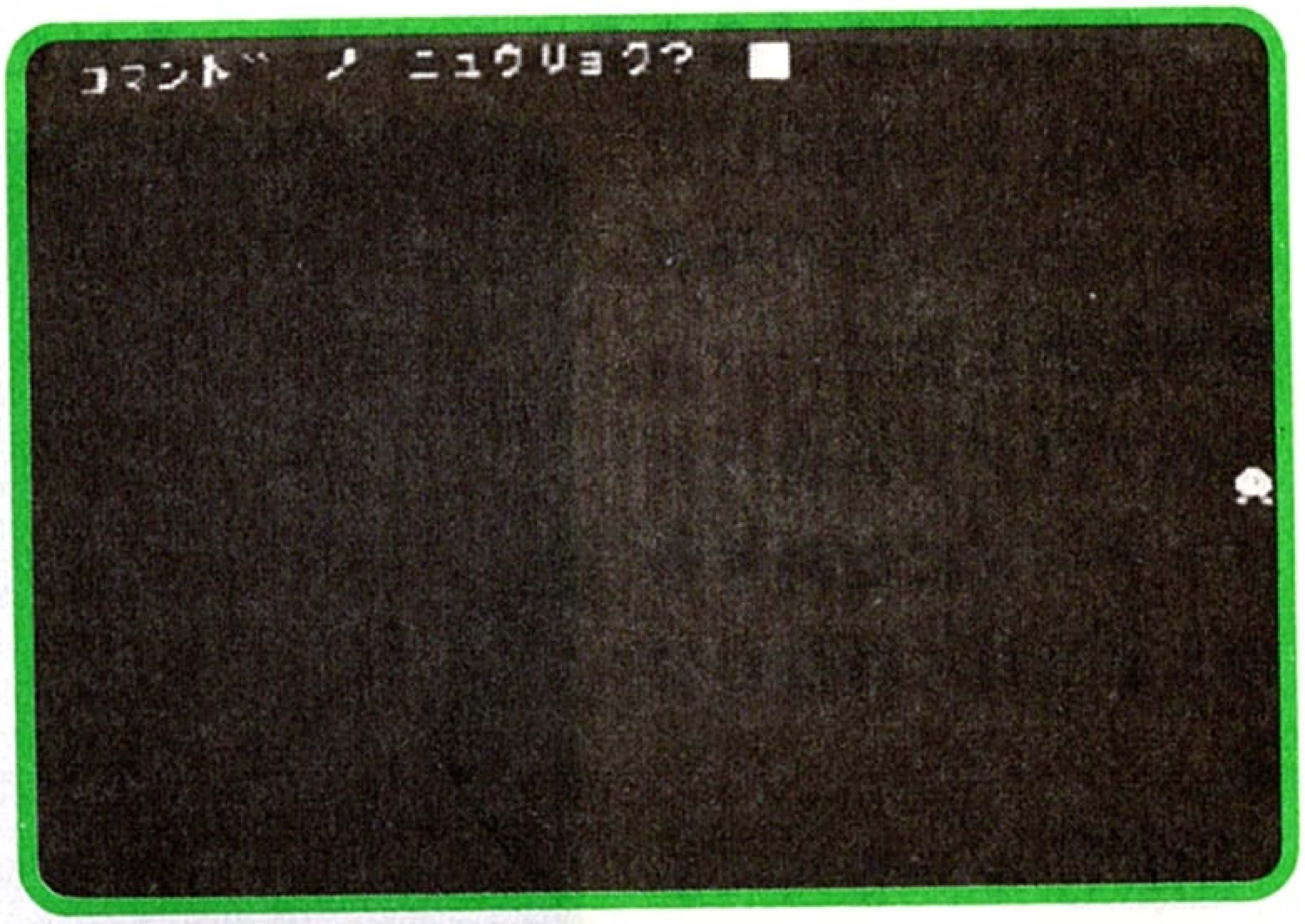
メモリ番地	マシン語	メモリ番地	マシン語
E000	21 50 E0	E022	3E FF
E003	11 00 38	E024	06 FF
E006	01 08 00	E026	10 FE
E009	CD 5C 00	E028	3D
E00C	06 F0	E029	20 F9
E00E	C5	E02B	C1
E00F	21 58 E0	E02C	10 E0
E012	11 00 1B	E02E	C9
E015	01 04 00		
E018	CD 5C 00	E050	3C 7E D7
E01B	3A 59 E0	E053	D7 FF 7E
E01E	3C	E056	24 C3 60
E01F	32 59 E0	E059	<u>00</u> 00 0F

このプログラムを実行しおわると、最後の行のE059番地の00（アンダーラインが引いてある箇所）がF0に変わります。そのまま実行させるとキャラクタは、移動しおわったところから、ふたたび移動しはじめるので、つぎのようにして、

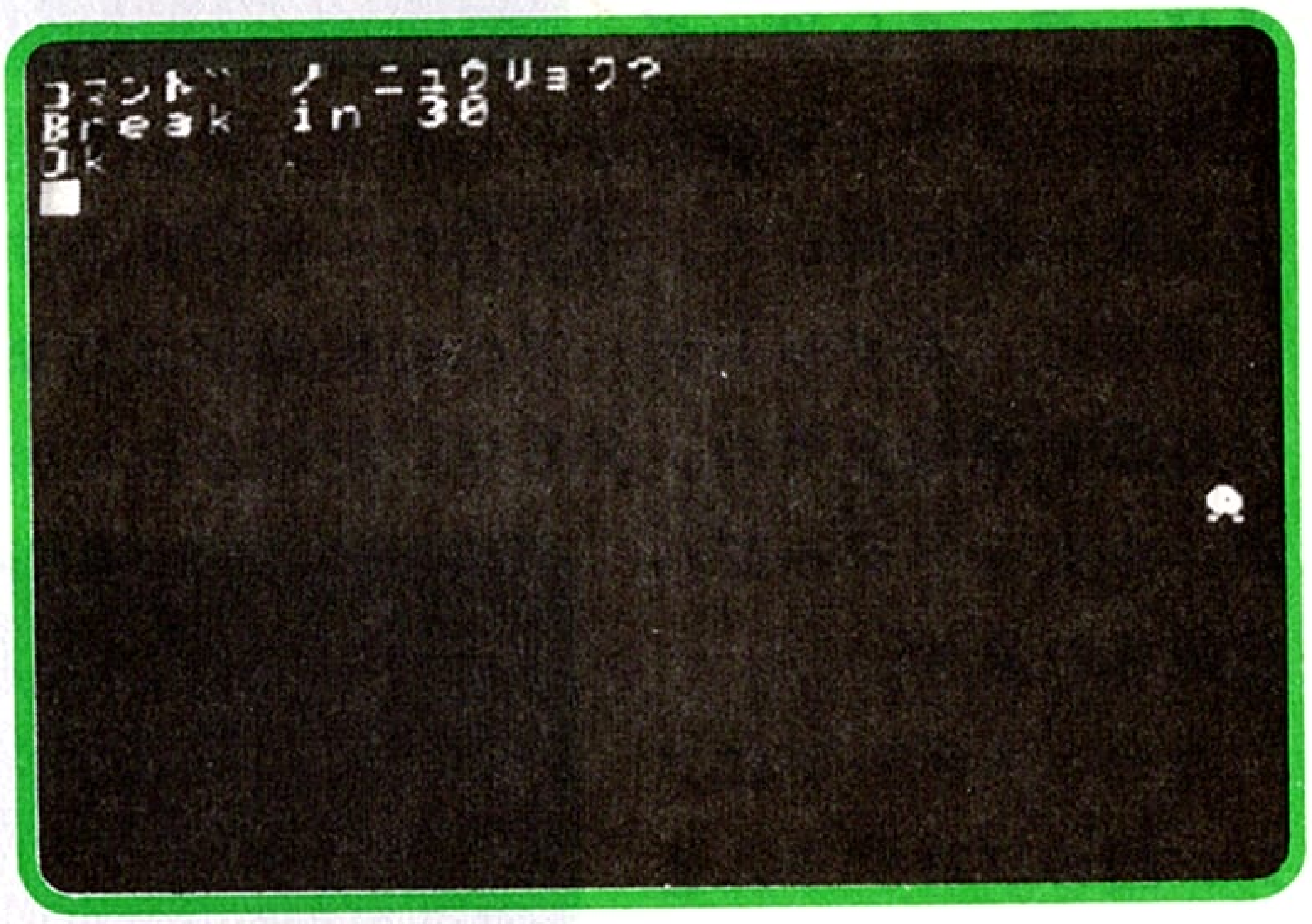
コマンド ノ リュウリョク? S
メモリバンチ ノ ニュウリョク? E059
E059 F0-00 ←――00に戻す

E059番地の値を00に戻してください。

では、カセットテープへの記録です。このプログラムを実行しおわったところから説明します。



右の写真が実行しおわった状態です。この状態は、マシン語モニタが実行されている状態ですから、BSAVEを入力することができません。したがって、[CTRL]キーと[STOP]キーを押して、マシン語モニタの実行を停止させます。



さて、このプログラムのファイル名です。ファイル名はなんでもよいのですが、キャラクタの移動なので「ｉｄｏ」とします。つぎは、この「ido」のプログラムが記憶されているメモリ番地です。

この場合プログラムは、Ｅ０００番地からＥ０５Ｂ番地までのメモリに記憶されています。

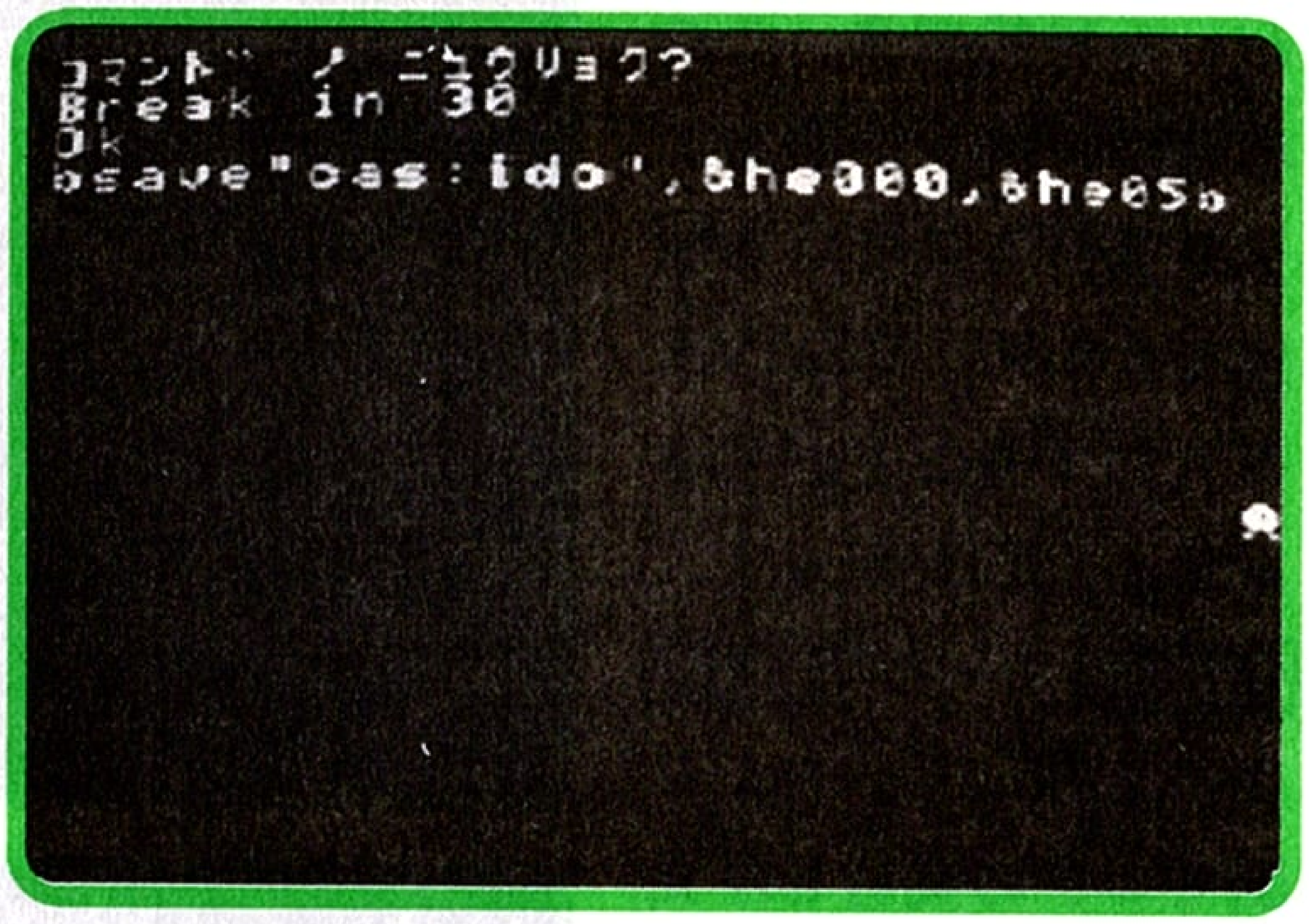
最初のメモリ番地はすぐわかるかもしれませんが、16進数に慣れていないと、最後のメモリ番地はわかりにくいかもしれません。最後の行に示されているメモリ番地は、Ｅ０５９です。Ｅ０５９番地のメモリの内容が最初の００ですから、つぎのように、

```
E059  00    00    0F
      ↑     ↑     ↑
      E059  E05A  E05B
```

Ｅ０５９、Ｅ０５Ａ、Ｅ０５Ｂと数えて判断すればよいのです。

したがって、マシン語のプログラムを記録するBSAVEには、つぎのように指定することになります。

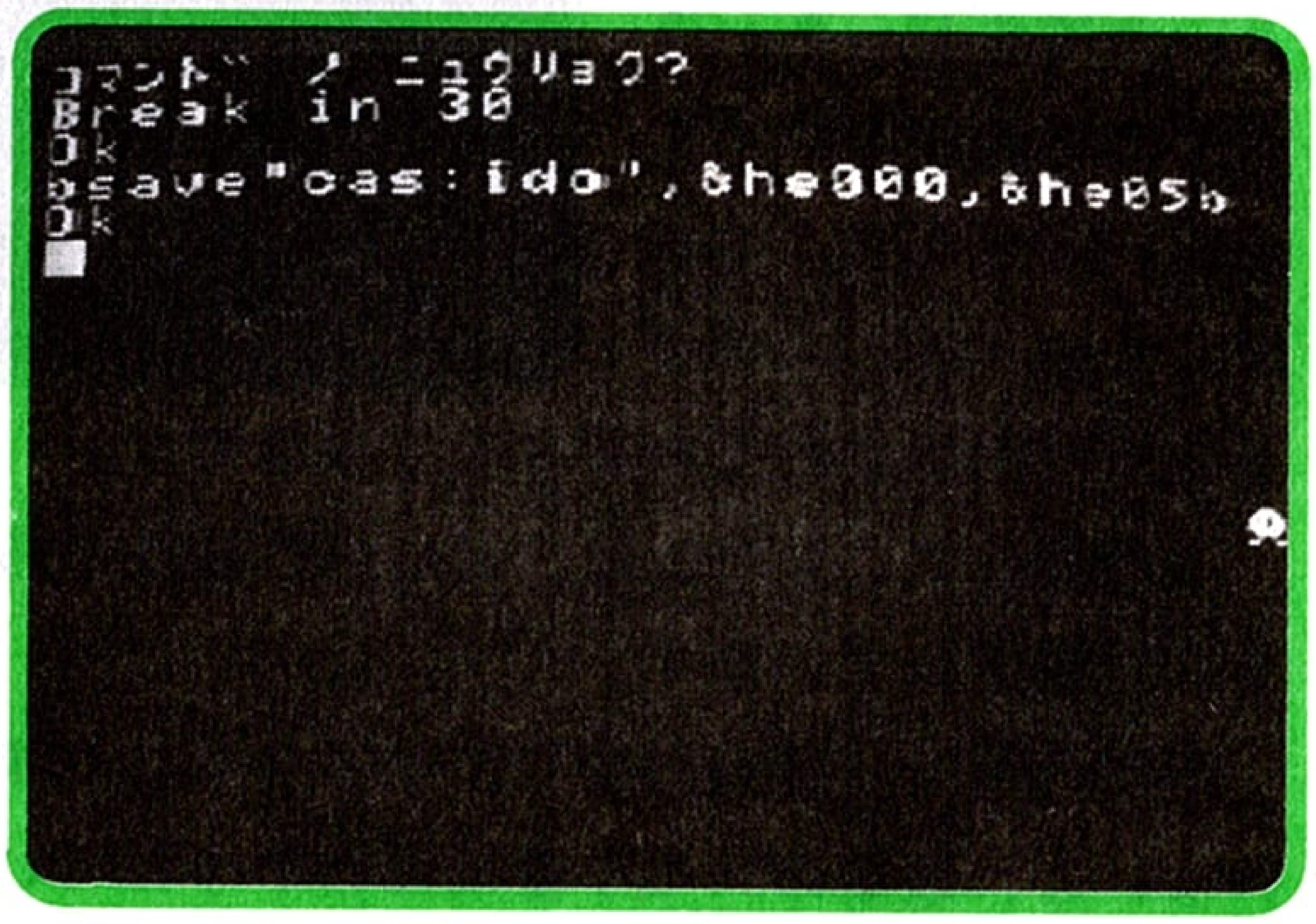
BSAVE "CAS：ido", &HE000, &HE05B



このように指定して入力したら（右の写真参照）、あとはベーシックのプログラムをカセットテープに記録するのとまったくおなじです。

RET キーを叩くまえに、データレコーダにカセットテープをセットします。そして、REC ボタンと PLAY ボタンを一緒に押して、RET キーを叩きます。

すると、カセットテープへの記録がはじまって、E000番地からE05B番地までの、メモリに記憶されているマシン語のプログラムが、カセットテープへ記録されます。



カセットテープにマシン語のプログラムを記録している間は、カーソルは表示されません。カセットテープへマシン語のプログラムを記録しおわると、右の写真に示すようにOKが表示されます。

カセットテープに記録したプログラムを読み込む

マシン語のプログラムを読み込むには、BLOAD

117頁で、ＢＳＡＶＥを使ってカセットテープに記録したマシン語のプログラムは、必要に応じて、カセットテープからＭＳＸパソコンのメモリに読み込んで使います。

ベーシックのプログラムを、カセットテープからＭＳＸパソコンのメモリのなかに読み込むコマンドは、CLOADです。マシン語のプログラムを、カセットテープからＭＳＸパソコンのメモリのなかに読み込むコマンドは、BLOADです。

BLOAD=マシン語をカセットテープから読み込む

BLOADは、つぎのような形をしています。

BLOAD " <ファイルスペック> "

<ファイルスペック>には、ＣＡＳとマシン語のプログラムをカセットテープに記録したときのファイル名を指定します。たとえば117頁で「ido」というファイル名で、マシン語のプログラムをカセットテープに記録しましたが、そのマシン語のプログラムをＭＳＸパソコンのメモリのなかに読み込むとすれば、

"CAS：ido"

ということになります。

BLOADは、BSAVEのときのような〈開始アドレス〉や、〈終了アドレス〉の指定をする必要がありませんから、BLOADは、

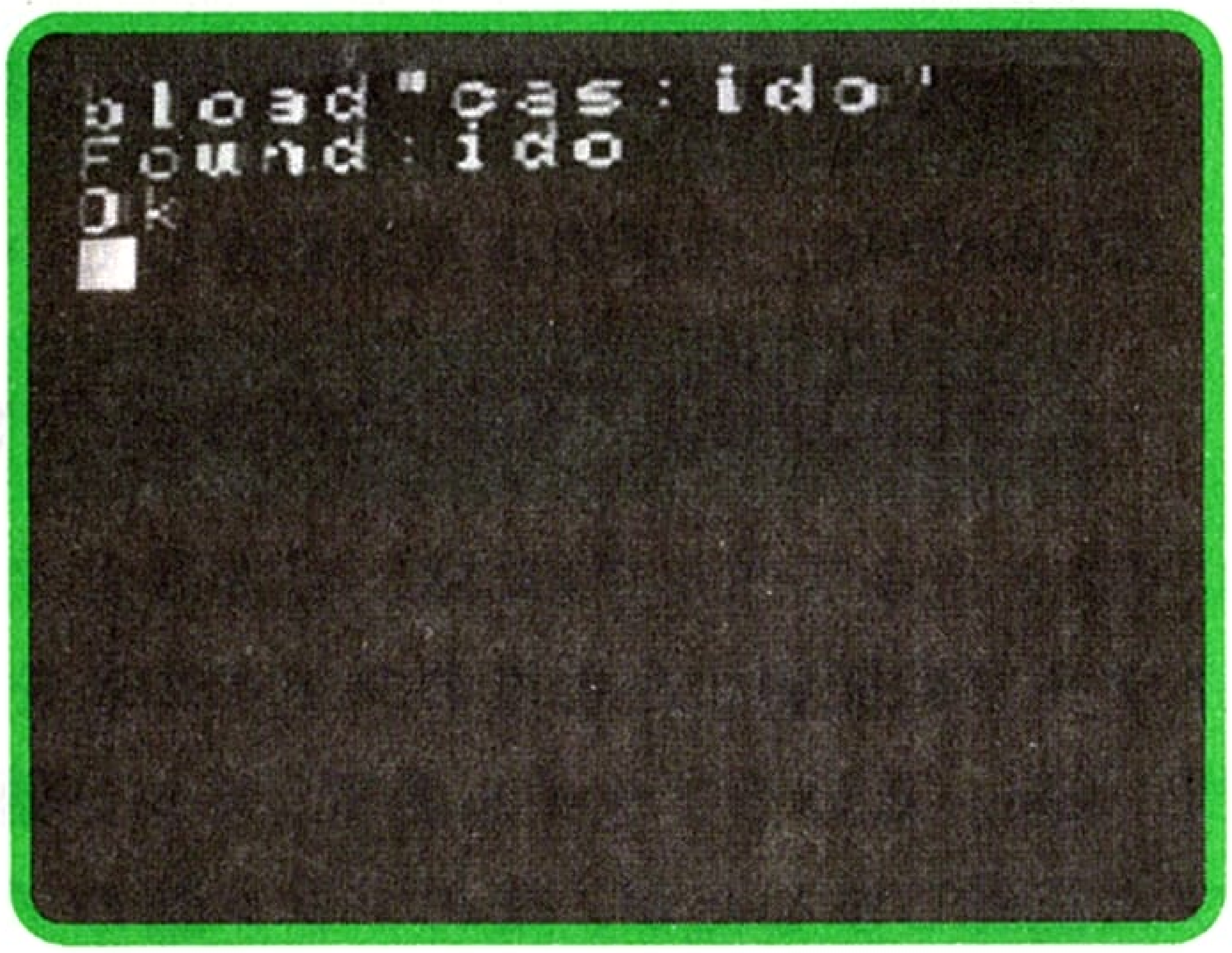
BLOAD "CAS：ido"

とするだけでよいのです。

このように指定して RET キーを叩くと、カセットテープに記録したときに指定したメモリ番地のメモリに、マシン語のプログラムが読み込まれて記憶されます。

では実際に、117頁でカセットテープに記録した「ido」のプログラムを、メモリに読み込むことにします。





まず、右の写真のように入力します。あとは、ベーシックのプログラムをメモリに読み込むのとおなじです。データレコーダの PLAY ボタンを押して RET キーを叩くと、入力した「ido」のファイル名を捜します。「ido」のまえに、ほかのマシン語のプログラムのファイル名があると、skipします。「ido」のファイル名を見つけだすと、右の写真に示すように、Found：idoを 表示して、「ido」のマシン語プログラムをメモリに読み込みはじめます。

マシン語のプログラムをメモリのなかに読み込んでいる最中は、カーソルは表示されません。マシン語のプログラムの読み込みが終了すると、OKが表示されます。

BLOADで、カセットテープからマシン語のプログラムをメモリに読み込むまえに、マシン語モニタをCLOADでメモリのなかに読み込んでおいてもかまいません。マシン語のプログラムを読み込むことで、マシン語モニタが破壊されることはないからです。

チェックサムとは

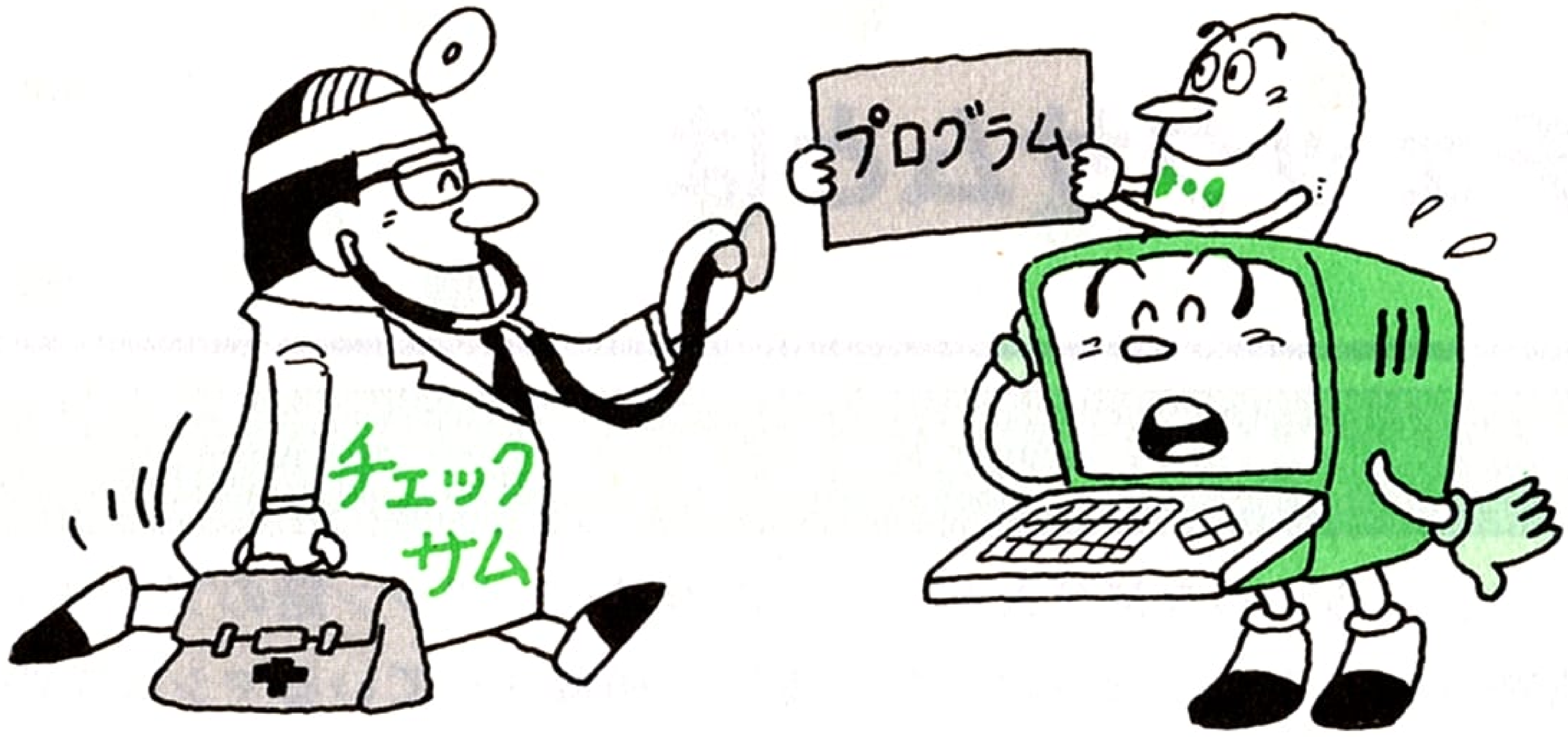
チェックサムは、入力ミスを調べる

今日、いろいろなＭＳＸパソコンの雑誌に、マシン語のプログラムが掲載されるようになりました。雑誌に掲載されているマシン語のプログラムに、つぎに示すように、各行のおわりにコロン（：）で区切られて、２桁の16進数が示されているのを、みたことがあるでしょうか。このコロンのあとに示されている２桁の16進数はマシン語ではなくて、**チェックサム**といいます。

```
E000 21 50 E0 11 00 38 01 08 :A3
E008 00 CD 5C 00 06 A0 C5 21 :B5
E010 58 E0 11 00 1B 01 04 00 :69
E018 CD 5C 00 3A 59 E0 3C 32 :0A
E020 59 E0 3A 58 E0 3C 32 58 :71
E028 E0 3E FF 06 FF 10 FE 3D :6D
E030 20 F9 C1 10 D9 C9       :8A
```

チェックサムは、check sumと書きます。checkは「調べる」、「照合する」という意味です。sumは「合計」、「総和」という意味、つまりチェックサムは、調べるための合計ということです。

では、何を調べるための合計なのでしょうか。私たちがマシン語のプログラムを入力したとき、入力したマシン語が間違いなく入力されているかどうかを調べるためです。



ただし、このチェックサムは、MSXパソコンが自動的につけてくれるわけではありません。たとえ、マシン語モニタを内蔵しているパソコンであってもつけてくれません。このチェックサムは、マシン語のプログラムを作った人が、そのマシン語のプログラムを使う人が入力ミスなどで困らないように、つけてくれているものなのです。

チェックサムのつけ方

マシン語のプログラムのチェックサムのつけ方には、いろいろあります。まえの頁に取りあげたマシン語のプログラムのチェックサムは**単純サム方式**といいます。単純サム方式は、各行のマシン語を足していって合計を求め、その合計の下２桁をチェックサムとしたものです。

つまり１行目の場合は、

E000　21　50　E0　11　00　38　01　08

とマシン語が並んでいます。マシン語は16進数ですから、16進数の足し算を行って、その合計を求めます。

21＋50＋E0＋11＋00＋38＋01＋08＝1A3

１行目の最初の16進数２１から最後の16進数０８までを合計すると、16進数で１Ａ３になります。合計１Ａ３の下２桁をとると、Ａ３になります。この下２桁のＡ３を１行目のチェックサムとして、コロン(：)のあとにおきます。

2行目の場合は、

　D008　00　CD　5C　00　06　A0　C5　21

ですから、まえのように足していきます。

　00+CD+5C+00+06+A0+C5+21=2B5

2行目の合計は16進数で2B5となりますから、まえのときとおなじように、2B5の下2桁B5をとります。そして、このB5を2行目のチェックサムとして、コロン（：）のあとにおきます。

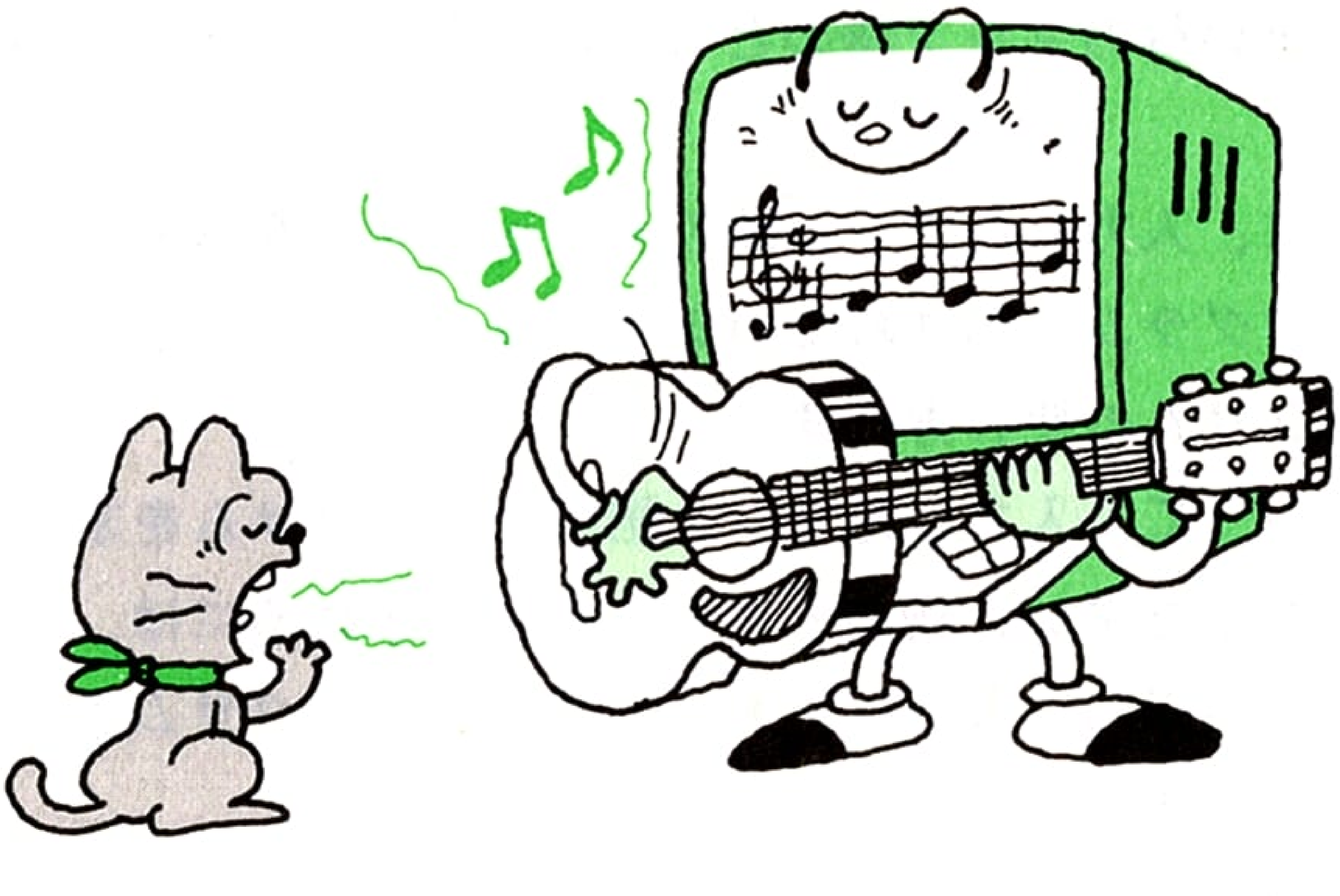
3行目以降のチェックサムは、1行目、2行目とおなじ方法でつけていきます。これが、単純サム方式によるチェックサムのつけ方です。

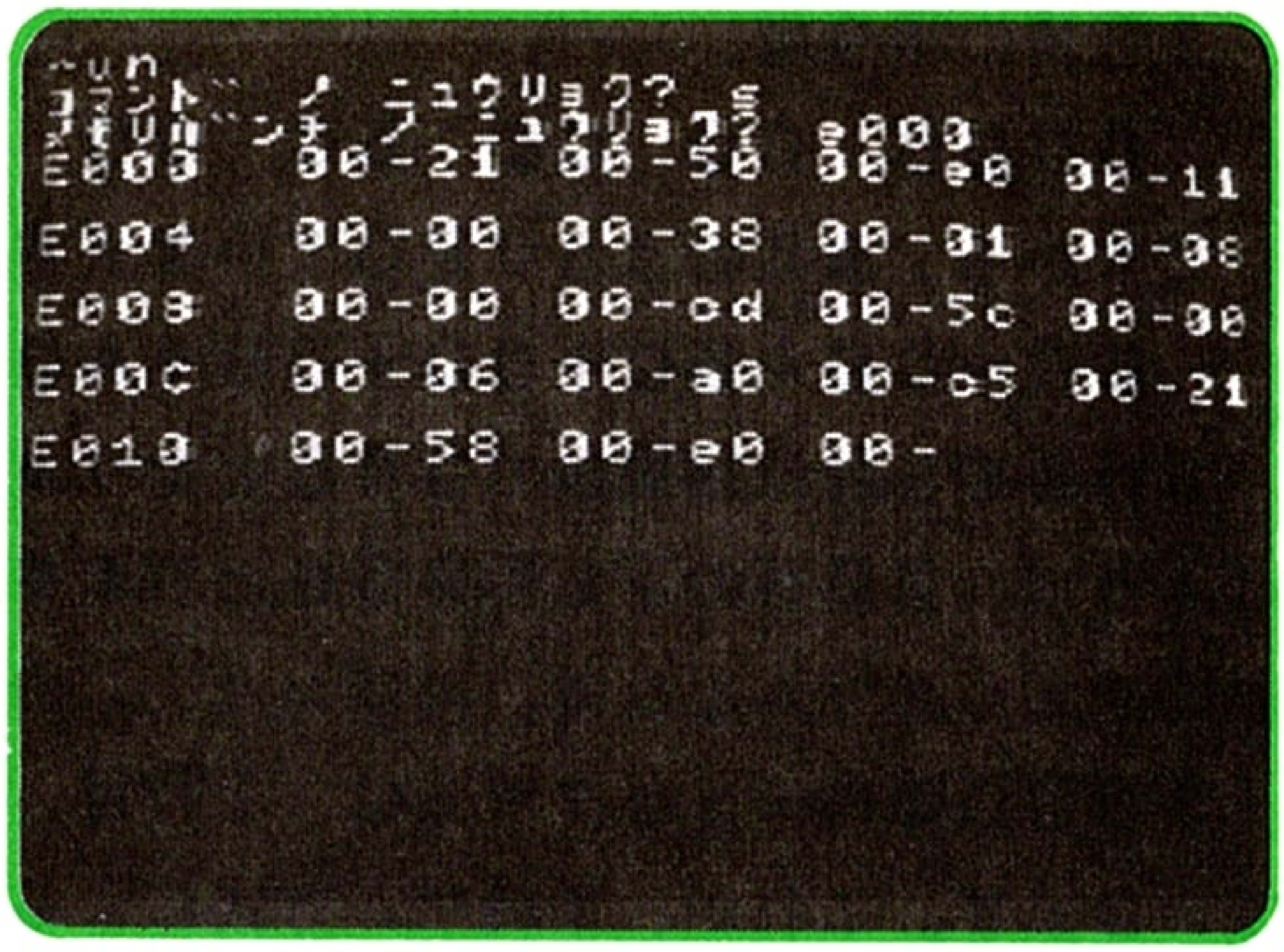
このようにチェックサムは、16進数でできているマシン語の合計を求めたものですから、マシン語のプログラムと一緒に入力してはいけません。

では、マシン語のプログラムについているチェックサムで、入力したマシン語のプログラムに入力ミスがあるかないかをどのようにして調べるのか、説明することにします。

チェックサムの使い方

まず、チェックサムのついたマシン語のプログラムを入力します。ここでは、まえの頁に示したマシン語のプログラムを入力することにします。ただし、このプログラムは、チェックサムを説明するためのものですから、実行しても何も表示されません。





さてマシン語のプログラムには、メモリ番地がＥ０００からつけられているので、Ｅ０００番地から右の写真のように入力していって、入力しおわったら、入力ミスがあるかないかチェックサムでチェックします。

ここで入力したような短いマシン語のプログラムでは入力ミスなどないかもしれませんが、長いマシン語のプログラムを間違いなく入力することはほとんどできません。マシン語のプログラムが長くなればなるほど、入力ミスがおこる確率は高くなるといってもよいのです。

入力したマシン語のプログラムに、入力ミスがあるかないかを調べるために、入力したマシン語のプログラムを画面に表示させて、もとのマシン語のプログラムと、ひとつひとつ照らし合せていたのでは頭が痛くなります。

そこで、入力したマシン語のチェックサムを求めます。そして求めたチェックサムと、もとのマシン語のプログラムについているチェックサムとを照らし合せます。両方のチェックサムが一致すれば、入力ミスがないということになります。チェックサムが一致しないところがあると、その行のマシン語のどこかに入力ミスがあるということになります。

このように、入力したマシン語のチェックサムを求めて、もとのマシン語のプログラムに示されているチェックサムと照らし合せる、これがチェックサムの使い方です。

ではどうやって、入力したマシン語のチェックサムを求めるのでしょうか。入力したマシン語のチェックサムを求めるには、プログラムが必要です。つぎに、入力したマシン語のチェックサムを求めるプログラムを示します。

```
10 INPUT "ｽﾀｰﾄｱﾄﾞﾚｽ";ST$
20 INPUT "ｴﾝﾄﾞｱﾄﾞﾚｽ";EN$
30 SA=VAL("&h"+ST$)
40 EA=VAL("&h"+EN$)
50 SN=0
60 FOR I=SA TO SA+7
70 SN=SN+PEEK(I)
80 NEXT I
90 IF SA>=EA THEN END
100 PRINT HEX$(SA);" - ";HEX$(SA+7)
   ;"   sum:";RIGHT$(HEX$(SN),2)
110 SA=SA+8
120 GOTO 50
```

チェックサムの入力と実行

うえに示したチェックサムを求めるプログラムは、さきに説明した横１行の合計を求める単純サム方式です。また、このチェックサムを求めるプログラムは、123頁のマシン語のプログラムに示したように、8個のマシン語のあとにチェックサムを表示するというものではなくて、各行の8個のマシン語の合計を求めて表示するというものです。このことは、あとで実行結果をみるとわかります。

ところで、まえの頁に示したように、マシン語のプログラムを入力しおわった状態は、まだ、マシン語モニタを実行している状態ですから、[CTRL] キーと [STOP] キーを一緒に押して実行を中止させます。中止させたら、チェックサムを求めるプログラムを入力します。

この場合ベーシックのプログラムは、自動的に８０００番地のメモリから入力されていきますから、マシン語モニタは破壊されてしまいます。

マシン語のチェックサムを求めるプログラムを入力しおわったら、RUNコマンドを入力します。すると、つぎのように、「スタートアドレス　ノ　ニュウリョク」のメッセージと？マークを表示します。

```
run
ｽﾀｰﾄｱﾄﾞﾚｽ ﾉ ﾆｭｳﾘｮｸ?
```

スタートアドレスは、マシン語のプログラムが記憶されている最初のメモリ番地です。この場合は、Ｅ０００ですから、

```
run
ｽﾀｰﾄｱﾄﾞﾚｽ ﾉ ﾆｭｳﾘｮｸ? e000
```

と入力します。スタートアドレスを入力したら、RETキーを叩きます。するとつぎに、「エンドアドレス　ノ　ニュウリョク」のメッセージと？マークを表示します。

```
run
ｽﾀｰﾄｱﾄﾞﾚｽ ﾉ ﾆｭｳﾘｮｸ? e000
ｴﾝﾄﾞｱﾄﾞﾚｽ ﾉ ﾆｭｳﾘｮｸ?
```

エンドアドレスは、マシン語のプログラムの最後のマシン語が記憶されているメモリ番地です。この場合は、最後のマシン語はＣ９で、Ｃ９はＥ０３５番地のメモリに記憶されていますから、エンドアドレスはＥ０３５となります。

```
run
ｽﾀｰﾄｱﾄﾞﾚｽ ﾉ ﾆｭｳﾘｮｸ? e000
ｴﾝﾄﾞｱﾄﾞﾚｽ ﾉ ﾆｭｳﾘｮｸ? e035
```

このようにエンドアドレスを入力してRETキーを叩くと、つぎの頁の実行結果に示すように、入力したマシン語のプログラムの各行ごとにチェックサムが画面に表示されます。

実行結果

```
run
ｽﾀｰﾄｱﾄﾞﾚｽ? e000
ｴﾝﾄﾞｱﾄﾞﾚｽ? e035
E000 - E007   sum:A3
E008 - E00F   sum:B5
E010 - E017   sum:69
E018 - E01F   sum:0A
E020 - E027   sum:71
E028 - E02F   sum:6D
E030 - E037   sum:8A
Ok
```

sumのあとにある16進数2桁が、チェックサムです。したがって、このチェックサムと、123頁のマシン語のプログラムについているチェックサムを、上から順に照らし合せます。この場合は、各行ともチェックサムは一致しますから、入力ミスはないということになります。

入力ミスがあるときは

では、あえて入力ミスを作って、入力ミスがあるときはどうなるかをみることにします。入力するマシン語のプログラムは、おなじものにします。

つぎに示すのが、新たに入力してチェックサムを求めたものです。このチェックサムと、123頁のマシン語のプログラムのチェックサムを、上から順に照らし合せていきます。

```
run
ｽﾀｰﾄｱﾄﾞﾚｽ? e000
ｴﾝﾄﾞｱﾄﾞﾚｽ? e035
E000 - E007    sum:A3
E008 - E00F    sum:65
E010 - E017    sum:69
E018 - E01F    sum:0A
E020 - E027    sum:71
E028 - E02F    sum:6C
E030 - E037    sum:8B
Ok
```

照らし合せると、1行目は一致しますから入力ミスはありません。2行目は123頁のマシン語のプログラムのチェックサムがＢ５、入力したマシン語のプログラムのチェックサムが６５で一致しませんから、入力ミスがあることになります。

ここで入力ミスがあることがわかっても、すぐに訂正にはかかれません。なぜならさきに説明したように、チェックサムのプログラムを入力したことによって、マシン語モニタが破壊されてしまっているからです。

そこで2行目のチェックサムの、左側に表示されているメモリ番地、

　Ｅ００８−Ｅ００Ｆ　　ｓｕｍ：６５

Ｅ００８とＥ００Ｆをメモしておきます。なぜメモリ番地をメモしておくのかは、あとでわかります。

３行目、４行目、５行目は一致するので、入力ミスはありません。６行目は一致しないので、入力ミスがありますから、

```
E028-E02F     sum:6C
```

メモリ番地Ｅ０２８とＥ０２Ｆをメモしておきます。

このように、全部のチェックサムの照合がおわったら、マシン語モニタを入力して、チェックサムが一致しない行の訂正を行います。

入力ミスの訂正

マシン語モニタを入力して実行させたら、メモリの内容を画面に表示させるＤコマンドと、さきにメモした２行目のメモリ番地Ｅ００８とＥ００Ｆを入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e008,e00f
```

チェックサムの違っているメモリ番地を入力

このようにしてRETキーを叩くと、メモリ番地Ｅ００８からＥ００Ｆまでのメモリの内容が、画面に表示されます。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e008,e00f
E008 00 CD 0C 00 06 A0 C5 21
```

0Cではなくて5C

このように表示させたら、123頁のマシン語のプログラムと照らし合せて、どこが違っているかを見つけます。

この場合は、左から３個目の０Ｃが５Ｃでなければなりません。そこで、０Ｃを５Ｃに訂正します。０Ｃが記憶されているメモリ番地がＥ００Ａ番地であるとわかるときは、つぎのように、メモリの内容を変更するＳコマンドと、０Ｃが記憶されているメモリ番地Ｅ００Ａを入力します。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e008,e00f
E008 00 CD 0C 00 06 A0 C5 21
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e00a ←訂正するメモリ番地を入力
E00A 0C-5C 00- ←RETキーを叩く
        正しいマシン語を入力
```

ＥＯＯＡ番地を入力して RET キーを叩くと、メモリ番地ＥＯＯＡを表示して、ＥＯＯＡ番地のメモリの内容ＯＣと－を表示しますから、正しいマシン語５Ｃを入力します。５Ｃを入力すると、つぎのメモリ番地ＥＯＯＢのメモリの内容ＯＯと－が表示されますから、そこで RET キーを叩きます。これで、2行目の訂正はおわりです。

ＯＣが記憶されているメモリ番地がわからないときは、

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e008,e00f
E008 00 CD 0C 00 06 A0 C5 21
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? s
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e008 ←最初のメモリ番地を入力
E008 00- CD- 0C-5C 00-  ←正しいマシン語5Cを入力
   スペースキーを叩く    RETキーを叩く
```

8個のマシン語の最初に表示されているメモリ番地を入力します。この場合はＥＯＯ８です。メモリ番地ＥＯＯ８を入力して RET キーを叩くと、メモリ番地の内容ＯＯを表示します。訂正する必要のないところは、スペースキーを叩いて送っていきます。訂正するマシン語ＯＣが表示されたら、正しいマシン語５Ｃを入力して、つぎのメモリ番地の内容が表示されたところで、RET キーを叩きます。

6行目の訂正の仕方も、おなじようにして行えばよいので、説明は省略します。

メモリ番地の読み方

メモリ番地は、8個おきに表示される

Dコマンドを使って、メモリに記憶させた内容を表示させると、つぎのように、各行の最初にメモリ番地が表示され、そのあとにマシン語が、8個（8バイト）ずつ表示されます。

```
run
ｺﾏﾝﾄﾞ ﾉ ﾆｭｳﾘｮｸ? d
ﾒﾓﾘﾊﾞﾝﾁ ﾉ ﾆｭｳﾘｮｸ? e000,e035
E000 21 50 E0 11 00 38 01 08
E008 00 CD 5C 00 06 A0 C5 21
E010 58 E0 11 00 1B 01 04 00
E018 CD 5C 00 3A 59 E0 3C 32
E020 59 E0 3A 58 E0 3C 32 58
E028 E0 3E FF 06 FF 10 FE 3D
E030 20 F9 C1 10 D9 C9 00 00
```

うえに示した実行結果は、メモリ番地E000からE035までの



メモリに記憶されているマシン語のプログラムを画面に表示させたものです。したがって、各行の最初にＥ０００、Ｅ００８、Ｅ０１０、Ｅ０１８というメモリ番地が表示されて、そのあとに8個(8バイト)ずつマシン語が表示されます。

この場合、各行の最初に表示されるメモリ番地は、どのメモリのメモリ番地かというと、メモリ番地のすぐあとに表示されたマシン語を記憶しているメモリのメモリ番地です。つまり、

1行目のＥ０００ ➡ ２１
2行目のＥ００８ ➡ ００
3行目のＥ０１０ ➡ ５８
4行目のＥ０１８ ➡ ＣＤ

ということです。

表示されるのは、最初のメモリ番地だけ

このように、各行の最初のマシン語を記憶しているメモリのメモリ番地は表示されますが、2番目からのマシン語を記憶しているメモリのメモリ番地は表示されません。したがって、2番目からのマシン語を記憶しているメモリのメモリ番地は、各行の最初に表示されるメモリ番地にもとづいて、数えていくことになります。

この場合は、1行目に表示されているメモリ番地はＥ０００で、そのあとに表示されたマシン語２１を記憶しているメモリのメモリ番地ですから、

2番目のマシン語５０を記憶しているメモリのメモリ番地はＥ００１
3番目のマシン語Ｅ０を記憶しているメモリのメモリ番地はＥ００２
4番目のマシン語１１を記憶しているメモリのメモリ番地はＥ００３
5番目のマシン語００を記憶しているメモリのメモリ番地はＥ００４
6番目のマシン語３８を記憶しているメモリのメモリ番地はＥ００５
7番目のマシン語０１を記憶しているメモリのメモリ番地はＥ００６
8番目のマシン語０８を記憶しているメモリのメモリ番地はＥ００７

というように数えていって、2行目の最初に示されているメモリ番

地Ｅ００８となります。

つぎに132頁に示したマシン語のプログラムの、メモリ番地の読み方をまとめて示します。

	E000	E001	E002	E003	E004	E005	E006	E007
E000	21	50	E0	11	00	38	01	08
	E008	E009	E00A	E00B	E00C	E00D	E00E	E00F
E008	00	CD	5C	00	06	A0	C5	21
	E010	E011	E012	E013	E014	E015	E016	E017
E010	58	E0	11	00	1B	01	04	00
	E018	E019	E01A	E01B	E01C	E01D	E01E	E01F
E018	CD	5C	00	3A	59	E0	3C	32
	E020	E021	E022	E023	E024	E025	E026	E027
E020	59	E0	3A	58	E0	3C	32	58
	E028	E029	E02A	E02B	E02C	E02D	E02E	E02F
E028	E0	3E	FF	06	FF	10	FE	3D
	E030	E031	E032	E033	E034	E035		
E030	20	F9	C1	10	D9	C9		

なお、Ｄコマンドで画面にマシン語を表示させると、１行に８個ずつ並ぶので、メモリ番地は８個おきに表示されますが、雑誌などのマシン語のプログラムでは、１行に16個（16バイト）ずつ並んでいることもあります。その場合メモリ番地は16個おきに示されています。

メモリ番地も16進数でつけられている

この場合、Ｅ０００からＥ００９と数えてきて、そのあとＥ００Ａ、Ｅ００Ｂ、Ｅ００Ｃ、Ｅ００Ｄ、Ｅ００Ｅ、Ｅ００Ｆと数えるのは、メモリ番地もマシン語とおなじように、16進数でつけられているからです。メモリ番地は、16進数４桁です。

メモリ番地＝16進数４桁

Ｚ－80命令表と使い方

Z-80命令表は、ハンドアセンブルで使う

Ｚ－80命令表は、ハンドアセンブル、つまり、アセンブリ言語で書いたプログラムを、自分でマシン語のプログラムになおすときに使います。142頁にＺ－80命令表を示します。この命令表で取りあげている命令は、ふつう使う基本的な命令だけで、８ビット、16ビット、それにジャンプ・コール・リターンの３つの表に分かれています。

まず、このうちの、８ビットの表の使い方から説明します。

8ビットの表を使ってのアセンブル

レジスタには、Ａ、Ｂ、Ｃ、Ｄ、Ｅ、Ｈ、Ｌの７つあります。この７つのレジスタのひとつひとつは、８ビットレジスタといって、８ビット（１バイト）の数値を記憶することができます。16進数でいうと、16進数２桁が８ビットですから、16進数２桁の数値を記憶することができるということになります。

８ビットレジスタ＝16進数２桁の数値を記憶

Ｚ－80命令表の８ビットの表は、この８ビットレジスタが使われているアセンブリ言語を、マシン語になおすときに使います。

では、つぎのアセンブリ言語のプログラムを、マシン語のプログラムになおしてみます。

```
LD   A, 25H ←――― Aレジスタに25Hを入れよ
LD   B, 30H ←――― Bレジスタに30Hを入れよ
ADD  A, B ←――― Aレジスタの内容にBレジスタの内容を足す
LD   (E032H), A ←Aレジスタの内容をE032番地のメモリに入れる
RET ←――― マシン語モニタに戻れ
```

１行目のＬＤ　Ａ，２５ＨのＡはＡレジスタのことで、８ビットレジスタですから、このアセンブリ言語をマシン語になおすときは、８ビットの表を使います。８ビットの表の左に、

ＬＤ　Ａ，×

この場合、×のところは２５Ｈという数値です。

ＬＤ　Ａ，２５Ｈ

⬇

ＬＤ　Ａ，×

×のところが数値のときは、表のうえの欄に示されているｎとなります。ｎは１バイトの数値を示していまから、ＬＤ　Ａ，×とｎが交わったところをみます。

３Ｅ

ｎ

ｎには、２５Ｈをそのままあてはめますから、マシン語は、

ＬＤ　Ａ，２５Ｈ←――アセンブリ言語

⬇

３Ｅ　２５←――マシン語

となります。ＬＤ　Ｂ，３０ＨのＢも、８ビットレジスタですから、おなじように８ビットの表を使います。表の左側に、

ＬＤ　Ｂ，×

とあります。この場合もまえの場合とおなじように、×のところは３０Ｈという数値です。

ＬＤ　Ｂ，３０Ｈ

⬇

ＬＤ　Ｂ，×

したがって、表のうえのｎとＬＤ　Ｂ，×が交わったところをみます。

０６

ｎ

ｎには、３０Ｈをそのままあてはめますから、マシン語は、

ＬＤ　Ｂ，３０Ｈ←アセンブリ言語

⬇

０６　３０←マシン語

となります。ＡＤＤ　Ａ，ＢのＡとＢは、まえとおなじ８ビットレジスタです。８ビットの表の左側をみると、

ＡＤＤ　Ａ，×

とあります。この場合、×のところはＢレジスタです。

ＡＤＤ　Ａ，Ｂ

⬇

ＡＤＤ　Ａ，×

×のところがＢレジスタのときは、表のうえの欄のＢのところをみます。このＢとＡＤＤ　Ａ，×が交わったところが、マシン語です。

ＡＤＤ　Ａ，Ｂ←アセンブリ言語

⬇

８０←マシン語

ＬＤ　(Ｅ０３２Ｈ)，ＡのＡは、まえとおなじ８ビットレジスタです。８ビットの表の左側に、

ＬＤ　(ｎｎ)，Ｘ

とあります。この場合は、×のところがＡレジスタですから、

ＬＤ　(Ｅ０３２Ｈ)，Ａ

⬋

ＬＤ　(ｎｎ)，Ｘ

表のうえのＡとＬＤ　(nn)，×とが交わったところをみます。

３２

ｎｎ

nnには、Ｅ０３２Ｈをあてはめます。ただし、16進数４桁の数値は、上位２桁のＥ０をあとに、下位２桁の３２をさきにしなければなりません。

上位２桁	下位２桁		下位２桁	上位２桁
Ｅ０	３２	➡	３２	Ｅ０

したがって、マシン語は、

LD　(E032H)，A←アセンブリ言語

⬇

32　32　E0←マシン語

となります。最後のRETは､143頁のジャンプ，コール，リターンの表を使います。表の左側をみると、

RET　×

とあります。この場合は、RETだけで×のところにはなにもありませんから、表のうえのUN　COND（無条件）と、RET　×が交わったところが、マシン語になります。

RET←アセンブリ言語

⬇

C9←マシン語

以上が、Z－80命令表の8ビットの表の使い方です。

16ビットの表を使ってのアセンブル

さきにA、B、C、D、E、H、Lの7つのレジスタは、8ビットレジスタと説明しました。この8ビットのレジスタを2つ組み合せて、16ビットのレジスタにすることができます。レジスタの組み合せ方は決まっていて、つぎのようになります。

BC　DE　HL←レジスタを2つ組み合せると16ビットレジスタ

このように、2つ組み合せたレジスタを**レジスタペア**といいます。

HL＝HLレジスタペア

レジスタを2つ組み合せてレジスタペアにすると、16ビットレジスタになりますから、16ビットの数値が記憶できるレジスタとなります。16ビットは、16進数2桁が8ビットですから、16進数4桁の数値となります。

16ビットレジスタ＝16進数4桁の数値を記憶

Z－80命令表の16ビットの表(143頁)は、この16ビットのレジスタが使われているアセンブリ言語をマシン語になおすときに使います。

では、つぎのアセンブラのプログラムを、マシン語のプログラムになおしてみます。

```
LD  HL, B050H ←── B050HをHLレジスタに入れよ
LD  BC, 3800H ←── 3800HをBCレジスタに入れよ
ADD  HL, BC ←── HLレジスタの内容にBCレジスタの内容を足せ
LD  (E010H), HL ←── HLレジスタの内容をE010番地のメモリに入れよ
RET ←── マシン語モニタに戻れ
```

ＬＤ　ＨＬ，Ｂ０５０ＨのＨＬは、ＨＬレジスタペアで16ビットのレジスタですから、16ビットの表を使います。表の左側に、

```
LD  HL, ×
```

とあります。この場合は、×のところはＢ０５０Ｈという16ビット（２バイト）の数値ですから、

```
LD  HL, B050H
      ↓
LD  HL, ×
```

表のうえの欄のnnとなります。nnは２バイト（16ビット）の数値を示しています。したがって、nnとＬＤ　ＨＬ，×が交わったところをみます。

```
21
nn
```

nnのところには、Ｂ０５０Ｈをあてはめますから、マシン語は、

```
LD  HL, B050H ←── アセンブリ言語
    ↓
21  50  B0 ←── マシン語
```

となります。ＬＤ　ＢＣ，３８００ＨのＢＣは、ＢＣレジスタペアで16ビットのレジスタですから、16ビットの表を使います。表の左側をみると、

```
LD  BC, ×
```

とあります。ＬＤ　ＢＣ，×の×のところは、まえの場合とおなじく16ビット（２バイト）の数値ですから、表のうえのnnとＬＤ　ＢＣ，

×が交わったところをみます。

　０１
　ｎｎ

nnのところには、３８００Ｈをあてはめますから、マシン語は、

　ＬＤ　ＢＣ，３８００Ｈ←アセンブリ言語
　　↓
　０１　００　３８←マシン語

となります。ＡＤＤ　ＨＬ，ＢＣのＨＬとＢＣは、まえとおなじです。したがって、16ビットの表の左側をみます。

　ＡＤＤ　ＨＬ，×

×のところはＢＣですから、表のうえのＢＣとＡＤＤ　ＨＬ，×が交わったところをみます。そこには、０９とありますから、

　ＡＤＤ　ＨＬ，ＢＣ←アセンブリ言語
　　↓
　０９←マシン語

となります。ＬＤ　（Ｅ０１０Ｈ），ＨＬのＨＬは、まえとおなじですから、16ビットの表の左側をみます。

　ＬＤ　（ｎｎ），×

この場合、ＬＤ　(nn)，×の×のところはＨＬですから、

　ＬＤ　（Ｅ０１０Ｈ），ＨＬ
　　↙
　ＬＤ　（ｎｎ），×

表のうえのＨＬと、ＬＤ　(nn)，×が交わったところをみます。

　２２
　ｎｎ

カッコのなかのnnには、Ｅ０１０Ｈをあてはめますから、

　ＬＤ　（Ｅ０１０Ｈ），ＨＬ←アセンブリ言語
　　↓
　２２　１０　Ｅ０←マシン語

となります。ＲＥＴは、８ビットも16ビットもおなじです。

Z－80命令表

8ビット

×	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	n	(nn)	I	R
LD A,×	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3E n	3A n n	ED 57	ED 5F
LD B,×	47	40	41	42	43	44	45	46			DD 46 d	FD 46 d	06 n			
LD C,×	4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d	0E n			
LD D,×	57	50	51	52	53	54	55	56			DD 56 d	FD 56 d	16 n			
LD E,×	5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d	1E n			
LD H,×	67	60	61	62	63	64	65	66			DD 66 d	FD 66 d	26 n			
LD L,×	6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d	2E n			
LD(HL),×	77	7(	71	72	73	74	75						36 n			
LD(BC),×	02															
LD(DE),×	12															
LD(IX+d),×	DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d						DD 36 d n			
LD(IY+d),×	FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d						FD 36 d n			
LD(nn),×	32 n n															
LD I,×	ED 47															
LD R,×	ED 4F															
ADD A,×	87	80	81	82	83	84	85	86			DD 86 d	FD 86 d	C6 n			
ADC A,×	8F	88	89	8A	8B	8C	8D	8E			DD 8E d	FD 8E d	CE n			
SUB ×	97	90	91	92	93	94	95	96			DD 96 d	FD 96 d	D6 n			
SBC A,×	9F	98	99	9A	9B	9C	9D	9E			DD 9E d	FD 9E d	DE n			
AND ×	A7	A0	A1	A2	A3	A4	A5	A6			DD A6 d	FD A6 d	E6 n			
XOR ×	AF	A8	A9	AA	AB	AC	AD	AE			DD AE d	FD AE d	EE n			
OR ×	B7	B0	B1	B2	B3	B4	B5	B6			DD B6 d	FD B6 d	F6 n			
CP ×	BF	B8	B9	BA	BB	BC	BD	BE			DD BE d	FD BE d	FE n			
INC ×	3C	04	0C	14	1C	24	2C	34			DD 34 d	FD 34 d				
DEC ×	3D	05	0D	15	1D	25	2D	35			DD 35 d	FD 35 d				

Z－80命令表

16ビット

×	BC	DE	HL	SP	IX	IY	AF	nn	(nn)
LD AF,×									
LD BC,×								0 1 n n	E D 4 B n n
LD DE,×								1 1 n n	E D 5 B n n
LD HL,×								2 1 n n	2 A n n
LD SP,×			F 9		D D F 9	F D F 9		3 1 n n	E D 7 B n n
LD IX,×								D D 2 1 n n	D D 2 A n n
LD IY,×								F D 2 1 n n	F D 2 A n n
LD(nn),×	E D 4 3 n n	E D 5 3 n n	2 2 n n	E D 7 3 n n	D D 2 2 n n	F D 2 2 n n			
PUSH ×	C 5	D 5	E 5		D D E 5	F D E 5	F 5		
POP ×	C 1	D 1	E 1		D D E 1	F D E 1	F 1		
ADD HL,×	0 9	1 9	2 9	3 9					
ADD IX,×	D D 0 9	D D 1 9		D D 3 9	D D 2 9				
ADD IY,×	F D 0 9	F D 1 9		F D 3 9		F D 2 9			
ADC HL,×	E D 4 A	E D 5 A	E D 6 A	E D 7 A					
SBC HL,×	E D 4 2	E D 5 2	E D 6 2	E D 7 2					
INC ×	0 3	1 3	2 3	3 3	D D 2 3	F D 2 3			
DEC ×	0 B	1 B	2 B	3 B	D D 2 B	F D 2 B			

ジャンプ，コール，リターン

×	UN COND	C	NC	Z	NZ	PE	PO	M	P	
J P ×， nn	C 3 n n	D A n n	D 2 n n	C A n n	C 2 n n	E A n n	E 2 n n	F A n n	F 2 n n	
JR ×, e	1 8 e−2	3 8 e−2	3 0 e−2	2 8 e−2	2 0 e−2					
JP(HL)	E 9									
JP(IX)	D D E 9									
JP(IY)	F D E 9									
CALL ×, nn	C D n n	D C n n	D 4 n n	C C n n	C 4 n n	E C n n	E 4 n n	F C n n	F 4 n n	
DJNZ e										1 0 e−2
RET ×	C 9	D 8	D 0	C 8	C 0	E 8	E 0	F 8	F 0	
RETI	E D 4 D									
RETN	E D 4 5									

入　　力

IN A, (n)	D B n
IN A,(C)	E D 7 8
IN B,(C)	E D 4 0
IN C,(C)	E D 4 8
IN D,(C)	E D 5 0
IN E,(C)	E D 5 8
IN H,(C)	E D 6 0
IN L,(C)	E D 6 8

Aレジスタと汎用レジスタ

レジスタは、ベーシックの変数とおなじ

レジスタは、CPUのなかにあって、データ（数値）を一時記憶しておくところです。レジスタは、いくつあるのかというと、

A　B　C　D　E　H　L

の7つあります。これに裏レジスタといわれるレジスタが7つありますが、裏レジスタはふつうの場合使いませんから、レジスタは、うえに示した7つといえます。

さてマシン語では、このレジスタをベーシックの変数とおなじように使います。つぎに、どんなレジスタが使われるかを示します。

```
LD   HL, E050H      ←HLレジスタにE050Hを入れる
LD   DE, 3800H      ←DEレジスタに3800Hを入れる
LD   BC, 0008H      ←BCレジスタに0008Hを入れる
CALL 005CH          ←005CHのルーチンをコール
LD   B, F0H         ←BレジスタにF0Hを入れる
PUSH BC             ←BCレジスタの内容をスタックに退避
LD   HL, E058H      ←HLレジスタにE058Hを入れる
LD   DE, 1B00H      ←DEレジスタに1B00Hを入れる
LD   BC, 0004H      ←BCレジスタに0004Hを入れる
CALL 005CH          ←005CHのルーチンをコール
LD   A, (E059H)     ←E059番地の内容をAレジスタに入れる
INC  A              ←Aレジスタの内容に1を足す
LD   (E059H), A     ←Aレジスタの内容をE059番地に入れる
LD   A, FFH         ←AレジスタにFFHを入れる
```

```
LD B, FFH       ←Bレジスタに FFH を入れる
DJNZ E026H      ←E026番地に戻って繰り返えす
DEC A           ←Aレジスタの内容から1を引く
JR NZ E024H     ←Zフラグが0でなければE024番地に戻る
POP BC          ←BCレジスタにスタックから値を戻す
DJNZ E00EH      ←E00E番地に戻って命令を繰り返えす
RET             ←マシン語モニタに戻る
```

このマシン語のプログラムは176頁で説明している、スプライト面の左から右にキャラクタを移動させるプログラムの一部分ですが、いろいろなレジスタが繰り返えし使われています。

マシン語のプログラムの横につけた説明をみるとわかると思いますが、1行目ではHLレジスタ、2行目ではDEレジスタ、3行目ではBCレジスタ、5行目ではBレジスタ、6行目ではBCレジスタというように、ほとんどの行にレジスタが使われています。このようにマシン語では、レジスタを使います。

レジスタは、8ビットのデータを記憶する

マシン語のプログラムでデータを入れて使うのは、さきに示したA、B、C、D、E、H、Lの7個のレジスタで、これら7個のレジスタは8ビットレジスタです。8ビットは16進数2桁ですから、

8ビット＝16進数2桁

したがって、つぎのように、

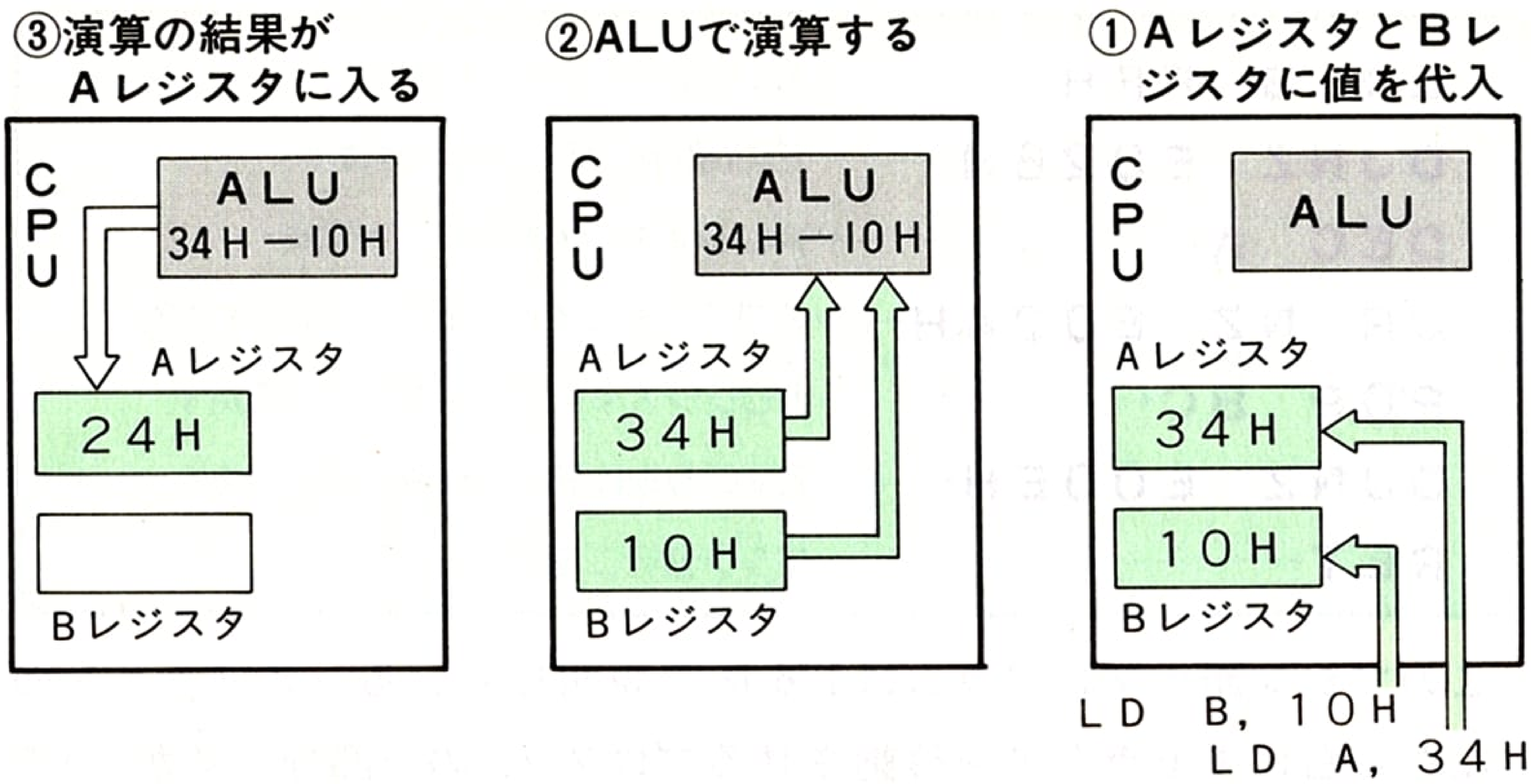
```
LD A, 49H       ←16進数2桁を入れることができる
```

16進数を2桁を入れることができますが、

```
LD A, 1A3BH     ←16進数4桁を入れることはできない
```

16進数4桁を入れることはできません。なぜなら、16進数4桁は16ビットだからです。

この7個の8ビットのレジスタのうち、Aレジスタは、**アキュムレータ**あるいは**累算器**（るいさんき）といわれます。



Aレジスタは、演算などで中心となるレジスタ

Aレジスタがアキュムレータとか累算器といわれるのは、演算（えんざん）などで中心となるレジスタだからです。

ここでは簡単な引き算でAレジスタの働きを説明します。

まず引かれる数をAレジスタに入れます。ここでは、引かれる数として３４H（10進数で52）を、Aレジスタに入れます。

```
LD　A，34H
```

このように引かれる数は、かならずAレジスタに入れることに決まっています。足し算の場合は、足される数をAレジスタに入れます。

引く数、あるいは足す数はどのレジスタに入れてもかまいません。ここでは、引く数を１０H（10進数で16）として、Bレジスタに入れることにします。

```
LD　B，10H
```

Bレジスタに引く数１０Hを入れたら、つぎはAレジスタの内容３４Hから、Bレジスタの内容１０Hを引きます。引き算の命令はSUB命令で、SUB命令のあとには、引く数が記憶されているBレジスタを置きます。

```
SUB　B
```

このように、SUB命令のあとにBレジスタだけをおくのは、SU

B命令で引き算を行うときは、Aレジスタの内容から引くと決まっているからです。

SUB　B命令を実行すると、146頁の図の②に示したように、ALUというところに送られて、引き算が行われます。引き算の結果の24H（16進数で36）は、図の③に示すように、Aレジスタに記憶されて残ります。

この場合は算術演算(さんじゅつえんざん)でしたが、論理演算(ろんりえんざん)、データの比較(ひかく)、データのメモリへの記憶、そのほかデータの入力や出力も、Aレジスタを使って行います。

汎用レジスタ

このように、Aレジスタにデータを入れて使うことができるといっても、その使い方は決まっていますが、そのほかのB、C、D、E、H、Lの6個のレジスタは、私たちが自由に使うことができるレジスタです。そのことから、この6個のレジスタを、**汎用(はんよう)レジスタ**といいます。

ところで、レジスタがどれもこれも、8ビットのレジスタであっては困ります。というのは、8ビットで表わすことができる数は255まででだからです。256以上の数は、16ビットで表わすことになりますから、16ビットを入れることができるレジスタが必要になります。16ビットは、16進数で4桁です。

16ビット＝16進数4桁

16ビットのデータを扱うときは、B、C、D、E、H、Lの6個の汎用レジスタをふたつ組み合せて、16ビットのレジスタにします。汎用レジスタの組み合せ方は、つぎのように決まっています。

BC　　DE　　HL←――レジスタの組み合せ方

このように、8ビットの汎用レジスタをふたつ組み合せた場合、**BCレジスタペア**とか、**HLレジスタ対**と呼びます。

このようにレジスタペアにして、16ビットのデータを処理する場合、中心になるレジスタペアは、HLレジスタペアです。ただし

16ビットのデータ処理で、HLレジスタペアが中心となるといっても、算術演算、16ビットのデータをメモリに記憶させるときぐらいです。

16ビットの演算は、HLレジスタペアが中心

ここでは、簡単な足し算でHLレジスタペアの働きについて説明します。8ビットの演算のときは、演算の中心となるAレジスタに入れました。16ビットの演算のときも足される数は、演算の中心となるHLレジスタペアに入れます。ここでは足される数として2F5BH(10進数で12123)を、HLレジスタペアに入れることにします。

LD HL, 2F5BH←――足される数をHLレジスタペアに入れる

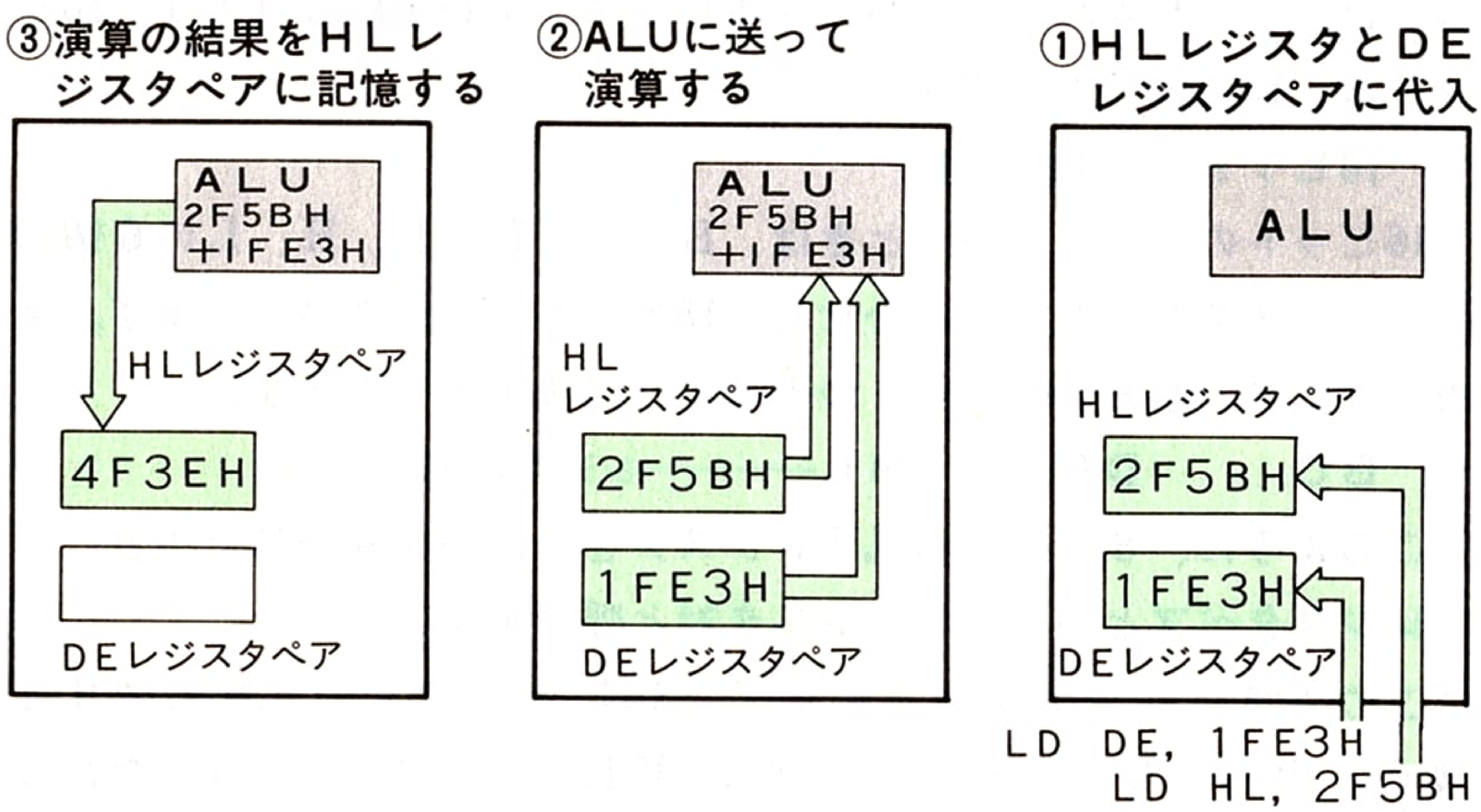
つぎは足す数を、そのほかのレジスタペアに入れます。ここでは足す数として、1FE3H (10進数で8163) をDEレジスタペアに入れることにします。

LD DE, 1FE3H←――足す数をDEレジスタペアに入れる

足される数をHLレジスタペアに、足す数をDEレジスタペアに入れたら、つぎはHLレジスタペアの内容2F5BHと、DEレジスタペアの内容1FE3Hを足します。足す命令は**ADD命令**です。

ADD HL, DE←HLレジスタペアの内容とDEレジスタペアの内容を足す

足し算の命令が実行されると、下の図②のようにALUに送られて



足し算が行われます。足し算の結果の４Ｆ３ＥＨ（10進数で20286)は、図③に示すように、ＨＬレジスタペアに記憶されて残ります。

ＨＬレジスタペアでは画面に表示することができないので、メモリに入れて記憶させることにします。ＨＬレジスタペアの内容４Ｆ３ＥＨは、Ｅ０１０番地のメモリに入れることにします。メモリに入れるには、Ｅ０１０番地をカッコで括って、ＨＬレジスタペアのまえにおきます。

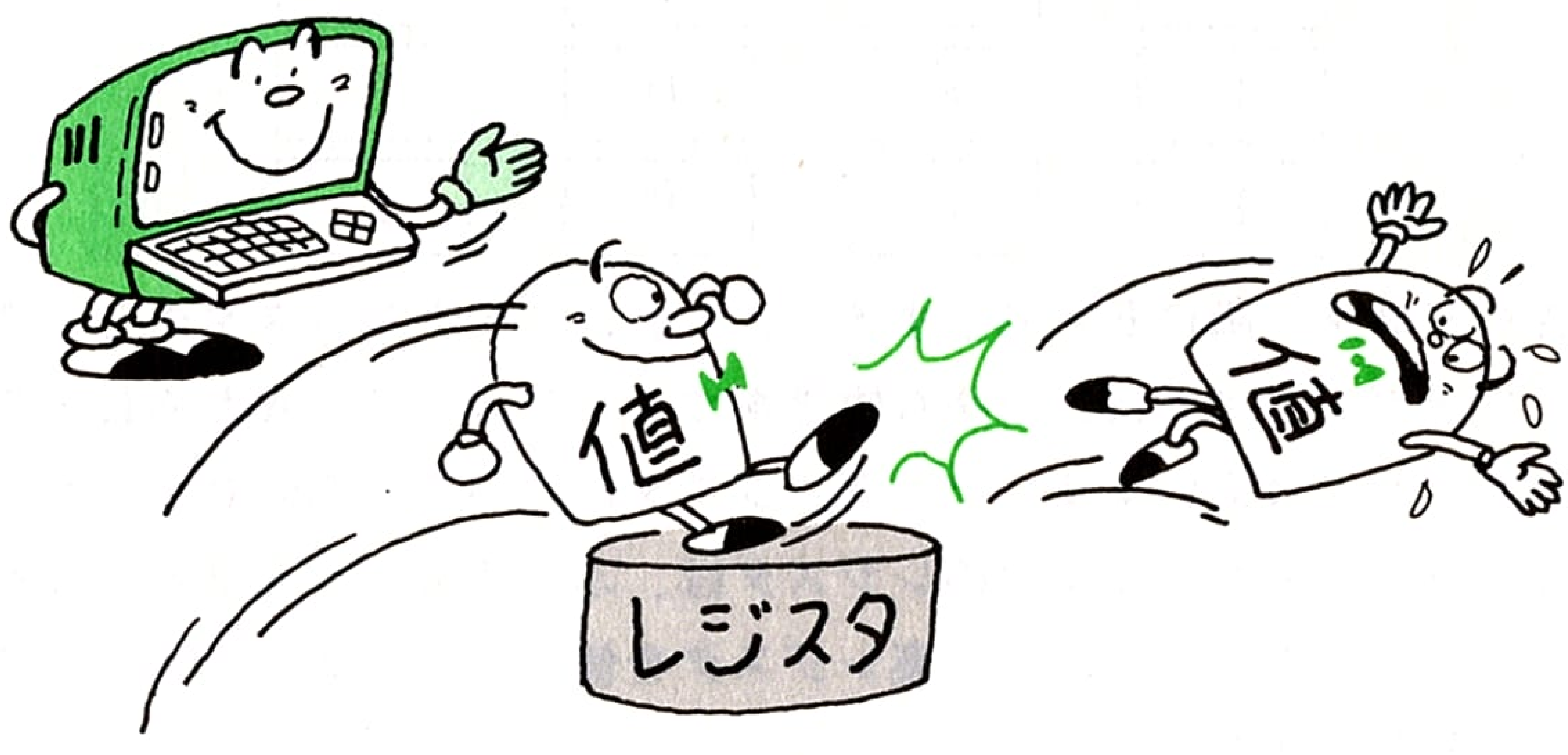
```
LD (E010H), HL
```
←HLレジスタペアの内容をメモリに入れて記憶させる

これで、ＨＬレジスタペアの内容４Ｆ３ＥＨは、Ｅ０１０番地のメモリに入れられて記憶されます。

レジスタは変数と同じで、値が変わる

このように、16ビットの演算のときはＢＣ、ＤＥ、ＨＬというようにレジスタペアにして使います。

この場合、ＢＣ、ＤＥ、ＨＬレジスタと、Ｂ、Ｃ、Ｄ、Ｅ、Ｈ、Ｌレジスタはおなじレジスタで、ただふたつ組み合せて使っているにすぎません。したがって、たとえばＢＣレジスタペアとして使っているとき、ＢレジスタとＣレジスタが、ＢＣレジスタとは違うレジスタだと勘違いして、ＢレジスタやＣレジスタに値を入れると、つぎに示すように、ＢＣレジスタに記憶されている値が変わってしまいます。



LD　BC, EF3EH ← BCレジスタペアにEF3EHを入れる

↓

BC　[EF3EH] ← BCレジスタの内容

↓

LD　B, 1CH ← Bレジスタに1CHを入れる

↓

BC　[1C3EH] ← BCレジスタの内容が変わる

レジスタはベーシックの変数とおなじで、新たに値を入れると、まえに入れた値が消えて、あとから入れた値が記憶されて残ります。

レジスタは8個しかありませんから、おなじレジスタを使うということが頻繁(ひんぱん)におこります。しかしこのように、まえに入れた値が消えてしまっては困ります。もちろん、もう必要のない値であれば消えてしまってもかまいませんが、まだその値が必要であるときは大変なことになります。そのようなときに使うのが、**スタック**です。スタックについては244頁で説明していますから、参照してください。

以上が、マシン語のプログラムでデータを入れて使うAレジスタと汎用レジスタです。つぎに、レジスタをまとめて示します。

表レジスタ		裏レジスタ	
A	F	A′	F′
B	C	B′	C′
D	E	D′	E′
H	L	H′	L′

Aレジスタの隣にFとありますが、これはフラグレジスタのことです。フラグレジスタは、特殊な働きをするレジスタで、AからLまでのレジスタとは違います。フラグレジスタについては234頁を参照してください。A′からL′までのレジスタは、ふつう**裏レジスタ**とよばれているレジスタです。ふつう裏レジスタを使うことはありませんから、説明は省略します。

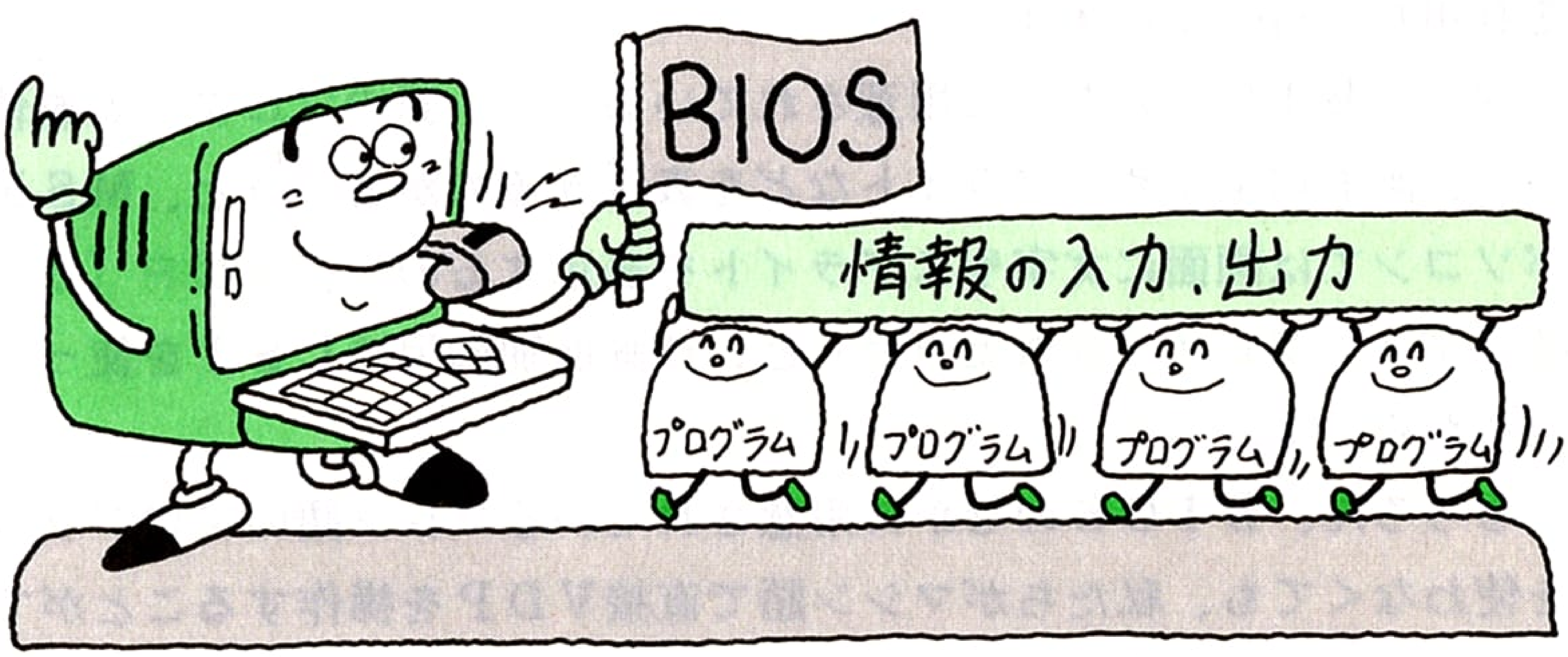
画面に文字やキャラクタを表示するルーチン

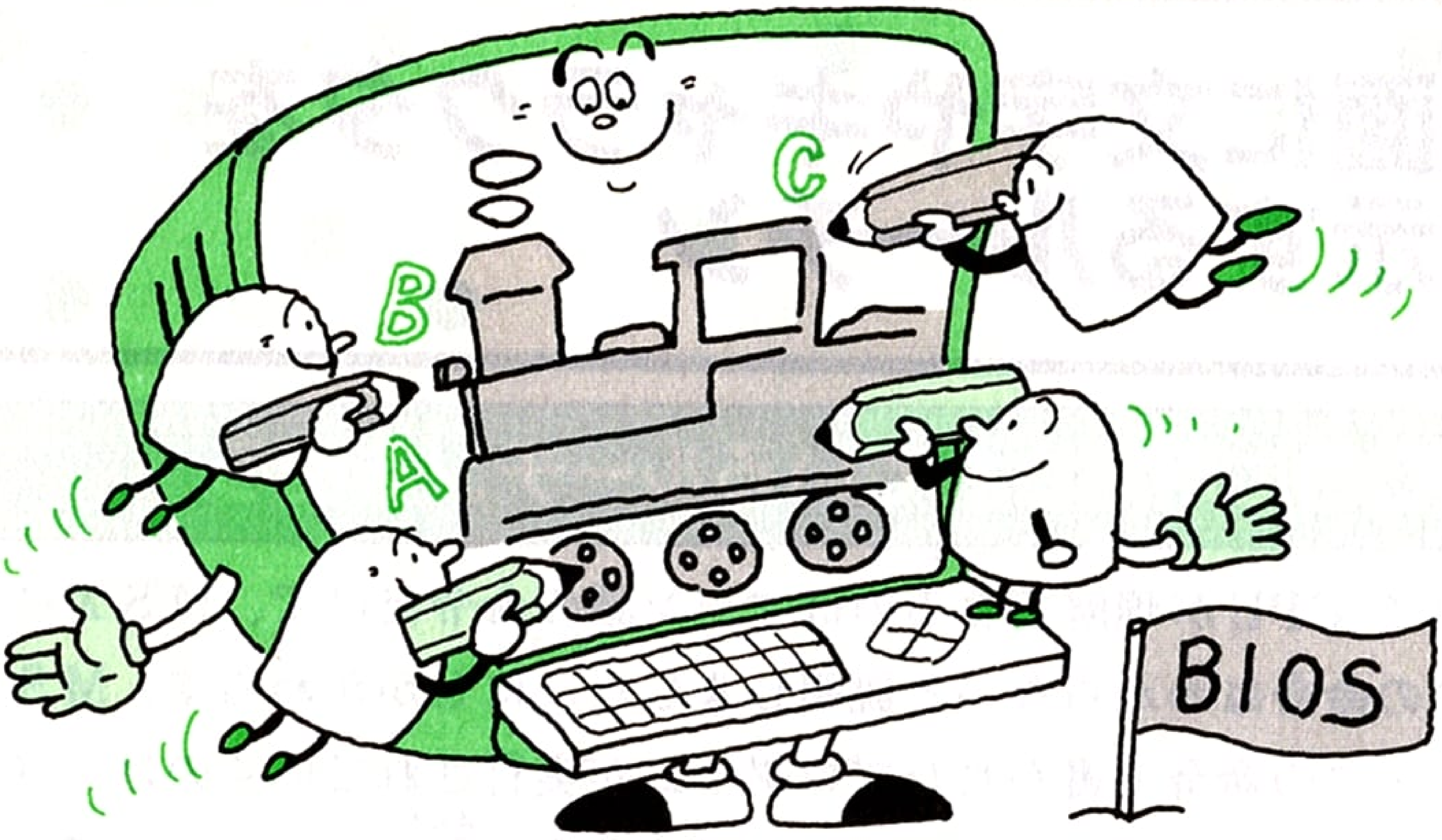
BIOSに用意されているルーチンを使う

12頁で、CPUが理解できるのはマシン語の命令だけで、MSXベーシックの命令はわからないと説明しました。にもかかわらず、MSXベーシックの命令で書かれたプログラムが実行されていくのは、ベーシックインタプリタによってマシン語の命令に翻訳(ほんやく)されるからであると説明しました。ベーシックインタプリタが、MSXベーシックの命令をマシン語の命令に翻訳することができるということは、いいかえれば、ベーシックインタプリタのなかに、MSXベーシックの命令に応じて働くことができる、いろいろなマシン語のプログラムがあるからなのです。たとえば、

PRINT "A"

で、テキスト画面にAという文字が表示されます。これは、ベーシックインタプリタのなかに、1文字をテキスト画面に表示する働きをするマシン語のプログラムがあるからです。





このようにベーシックインタプリタには、ＭＳＸベーシックに応じて働くいろいろなマシン語のプログラムがありますが、このうちの特に情報を入力したり、出力したりするマシン語のプログラムの集まりをＢＩＯＳ（バイオス）と呼んでいます。

BIOS=入出力に関するマシン語プログラムの集り

ＢＩＯＳとは、Basic Input Output Systemの略です。ふつうＭＳＸパソコンの場合、マシン語でテキスト画面に文字などを表示したり、スプライト画面にスプライト（キャラクタ）を表示したりするときは、このＢＩＯＳのなかに用意されているマシン語のプログラムを利用して行っています。

なぜ、ＢＩＯＳのなかに用意されているマシン語のプログラムを使って、画面に文字やスプライトなどを表示するのかというと、ＭＳＸパソコンでは画面に文字やスプライトを表示するのに、ＶＤＰ（ビデオ・ディスプレイ・プロセッサ）という画面制御用のＬＳＩを使っているからです。

もちろん、ＢＩＯＳのなかに用意されているマシン語のプログラムを使わなくても、私たちがマシン語で直接ＶＤＰを操作することができますが、特別な場合を除いて、私たちがマシン語で直接ＶＤＰを操

作するようなことはしません。

なお、ＲＯＭに内蔵されているマシン語のプログラムのひとつひとつは、サブルーチン、あるいはルーチンなどとよばれています。

VDPに対する出力ルーチン

ＶＤＰは、さきに説明したように、画面に文字やスプライトなどを表示するＬＳＩです。このＶＤＰに文字やスプライトなどを出力することで、文字やスプライトなどが画面に表示されます。したがって、ＶＤＰを操作するいろいろなマシン語のプログラム、つまりルーチンがＢＩＯＳのなかに用意されていますが、ここでは文字やスプライトなどを画面に表示するときに使う、ふたつのルーチンだけ取りあげて、示します。つぎに示すふたつのルーチンを知っていれば、ふつうの場合、画面への表示は十分だからです。

メモリ番地	ルーチンの名称	ルーチンの働きと操作
００５Ｃ	ＬＤＩＲＶＭ	メモリからVRAMへブロック転送する ＨＬレジスタに転送もとの先頭番地 ＤＥレジスタに転送さきの先頭番地 ＢＣレジスタに転送するデータの長さ 以上の３つを指定する
００Ａ２	ＣＨＰＵＴ	テキスト画面に指定した文字を表示する Ａレジスタに文字を入れる

うえに示した００５Ｃ番地からのルーチンは、スプライト面にスプライトを表示するときに使います。００Ａ２番地からのルーチンは、テキスト画面に文字などを表示するときに使います。

うえに示した、００５Ｃ番地のルーチンを使ったスプライト面へのスプライトの表示させ方については164頁を、００Ａ２番地のルーチンを使ったテキスト画面への文字の表示のさせ方については、155頁を参照してください。

効果音を作り出すルーチン

BIOSに用意されているルーチンを使う

153頁で、テキスト画面に文字を、スプライト面にスプライトを表示するために、ＢＩＯＳに用意されたルーチンを示しましたが、画面に文字やスプライトを表示するばかりでなく、効果音(こうかおん)などの音を作りだすことも必要です。

ＭＳＸパソコンの場合、文字やスプライトを画面に表示するのにＶＤＰというＬＳＩを使っていますが、効果音などの音を作りだすために、**ＰＳＧ（プログラマブル・サウンド・ジェネレータ）**というＬＳＩを使っています。ＰＳＧをマシン語で直接操作して、効果音などの音を作りだすこともできますが、ＢＩＯＳに、ＰＳＧを操作するためのルーチンが用意されています。このルーチンを使うと、簡単に効果音などの音を作りだすことができます。

ＰＳＧに対するルーチンは、ＢＩＯＳにいくつか用意されていますが、効果音などの音を作りだすルーチンとしては、つぎに示すルーチンがあります。

メモリ番地	ルーチンの名称	ルーチンの働きと操作
００９３	ＷＲＴＰＳＧ	**ＰＳＧのレジスタに値を出力する** ＡレジスタにＰＳＧレジスタの番号を入れる Ｂレジスタに出力する値を入れる

テキスト画面に文字などを表示する

CALL命令でルーチンを呼び出す

152頁で、ＭＳＸパソコンはＶＤＰというＬＳＩを使って画面表示を行っていると説明しました。そして、ＶＤＰは直接操作するのではなくて、ＢＩＯＳに用意されているルーチンを使って、テキスト画面に文字などを表示すると説明しました。

ここでは、ＢＩＯＳに用意されているルーチンを使って、テキスト画面へ文字などの表示のさせ方について説明することにします。

ＢＩＯＳに用意されているルーチンを使うには、ＣＡＬＬ命令を使います。ＣＡＬＬは、英語のＣＡＬＬ（コール）で、「呼ぶ」という意味で、CALL命令はサブルーチンにジャンプさせるときに使います。

ＣＡＬＬ命令＝サブルーチンを処理するときに使う命令

テキスト画面に文字などを表示するルーチンは、153頁に示したように、００Ａ２番地から内蔵されています。００Ａ２番地というルーチンの先頭のメモリ番地を、エントリーアドレスといいます。

ルーチンの先頭のメモリ番地＝エントリーアドレス

ＣＡＬＬ命令でルーチンを使うときは、ＣＡＬＬ命令のあとにエントリーアドレスを書きます。

ＣＡＬＬ　〈エントリーアドレス〉

この場合は、００Ａ２番地のルーチンを使うわけですから、

ＣＡＬＬ　００Ａ２Ｈ

となります。

ＣＡＬＬ　００Ａ２Ｈだけでは、テキスト画面になにも表示されません。153頁の００Ａ２番地のルーチンの項をみると、Ａレジスタに文字を入れるとなっています。つまりＡレジスタに、テキスト画面に

表示する文字をキャラクタコードで入れるということです。ここではテキスト画面に♠を表示することにします。したがって、162頁のキャラクタコード表をみて、♠のキャラクタコードを探します。♠のキャラクタコードは８０Ｈですから、つぎのようになります。

```
LD  A, 80H
```

さて、ＣＡＬＬ　００２ＡＨとＬＤ　Ａ，８０Ｈの、どちらをさきにするかです。Ａレジスタにテキスト画面に表示するキャラクタコードを入れてから、００Ａ２番地のルーチンをコールします。

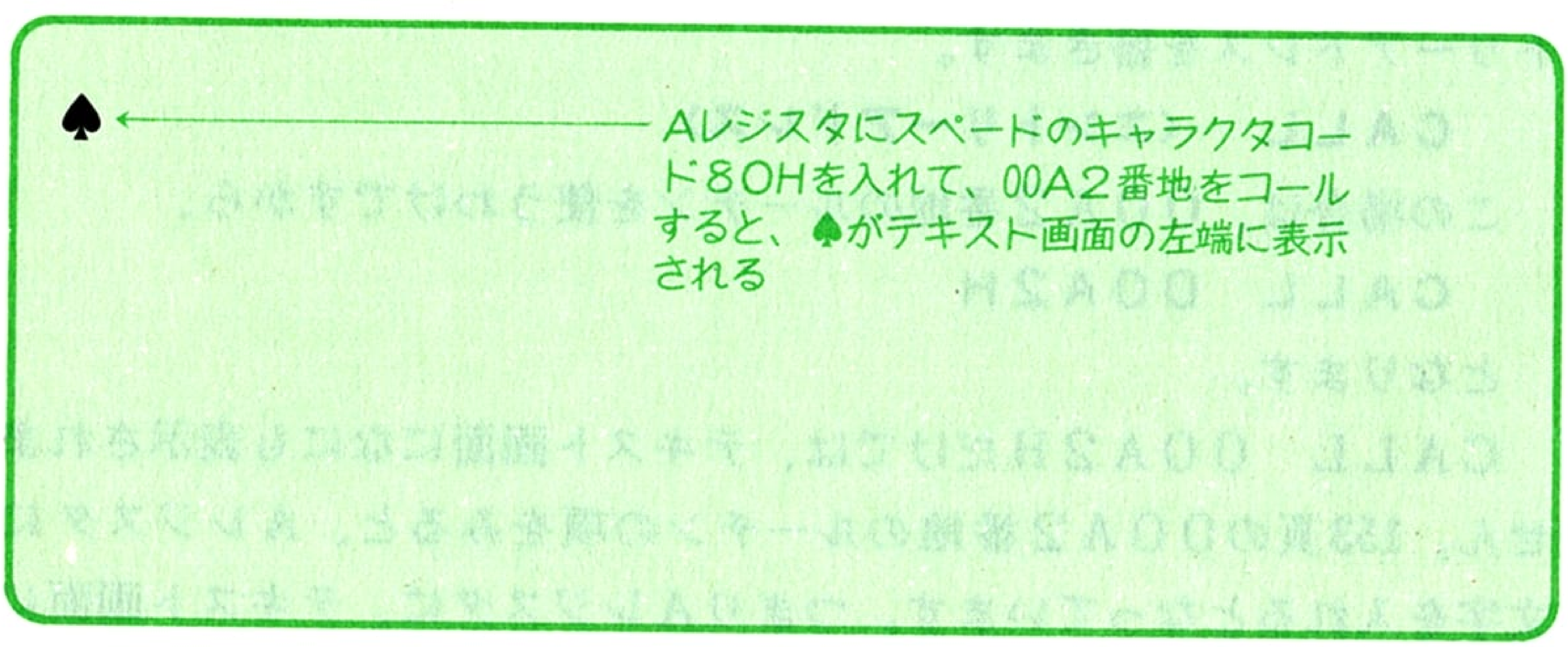
```
LD  A, 80H    ←――♠のキャラクタコードをAレジスタに入れる
CALL  00A2H   ←――00A2番地からのルーチンをコールする
RET           ←――マシン語モニタに戻す
```

アセンブリ言語のマシン語へのなおし方については、136頁を参照してください。ＣＡＬＬ命令は、ＲＥＴ命令とおなじように、ジャンプ・コール・リターンの表をみてマシン語になおします。

つぎにマシン語になおしたプログラムを示します。

メモリ番地	マ シ ン 語	アセンブリ言語
E000	3E 80	LD A, 80H
E002	CD A2 00	CALL 00A2H
E005	C9	RET

Ｇコマンドで、このマシン語のプログラムを実行させると、つぎの図のようにスペードが画面の左端上に表示されます。



指定した位置に文字などを表示するには

Ａレジスタに、テキスト画面に表示する文字のキャラクタコードを入れて、００Ａ２番地をコールすると、テキスト画面の左端上に表示されました。このように表示させるだけでなく、テキスト画面のある位置に文字を表示させることもあるので、つぎに、表示させる位置を指定して、文字をテキスト画面に表示させることにします。

テキスト画面に表示する位置を指定するには、行座標(ぎょうざひょう)（垂直方向(すいちょくほうこう)）と列座標(れつざひょう)（水平方向）を決めます。たとえば、Ｗを、行座標０ＣＨ(10進数で12）と列座標１０Ｈ（10進数で16）の位置に表示すると決めたとします。

そうしたら、まず行座標の０ＣＨをＡレジスタに入れます。

```
LD  A, 0CH  ← 行座標0CHをAレジスタに入れる
```

行座標の０ＣＨをＡレジスタに入れたら、つぎに、Ａレジスタの内容０ＣＨを、Ｆ３ＤＣ番地のメモリに記憶させます。

```
LD  (F3DCH), A  ← Aレジスタの内容0CHをF3DC番地のメモリに記憶させる
```

Ａレジスタの内容をメモリに記憶させるときは、このように、記憶させるメモリ番地をカッコで括って、Ａレジスタのまえにおけばよいのです。

つぎは、おなじように列座標の１０ＨをＡレジスタに入れます。

```
LD  A, 10H  ← 列座標10HをAレジスタに入れる
```

Ａレジスタに入れた列座標の１０Ｈは、Ｆ３ＤＤ番地のメモリに記憶させます。

```
LD  (F3DDH), A  ← Aレジスタの内容10HをF3DD番地のメモリに記憶させる
```

行座標、列座標を指定する値を記憶させるメモリ番地は、つぎのように決まっています。

行座標＝Ｆ３ＤＣ番地のメモリ

列座標＝Ｆ３ＤＤ番地のメモリ

このメモリ番地以外のメモリに行座標、列座標を指定する値を記憶させたときは、行座標、列座標を指定したことにはなりません。

行座標と列座標の値を決められたメモリ番地のメモリに記憶させたら、Aレジスタに、表示するWのキャラクタコードを入れます。

Wのキャラクタコードは、57Hです。

```
LD A, 57H
```
←Wのキャラクタコード57HをAレジスタに入れる

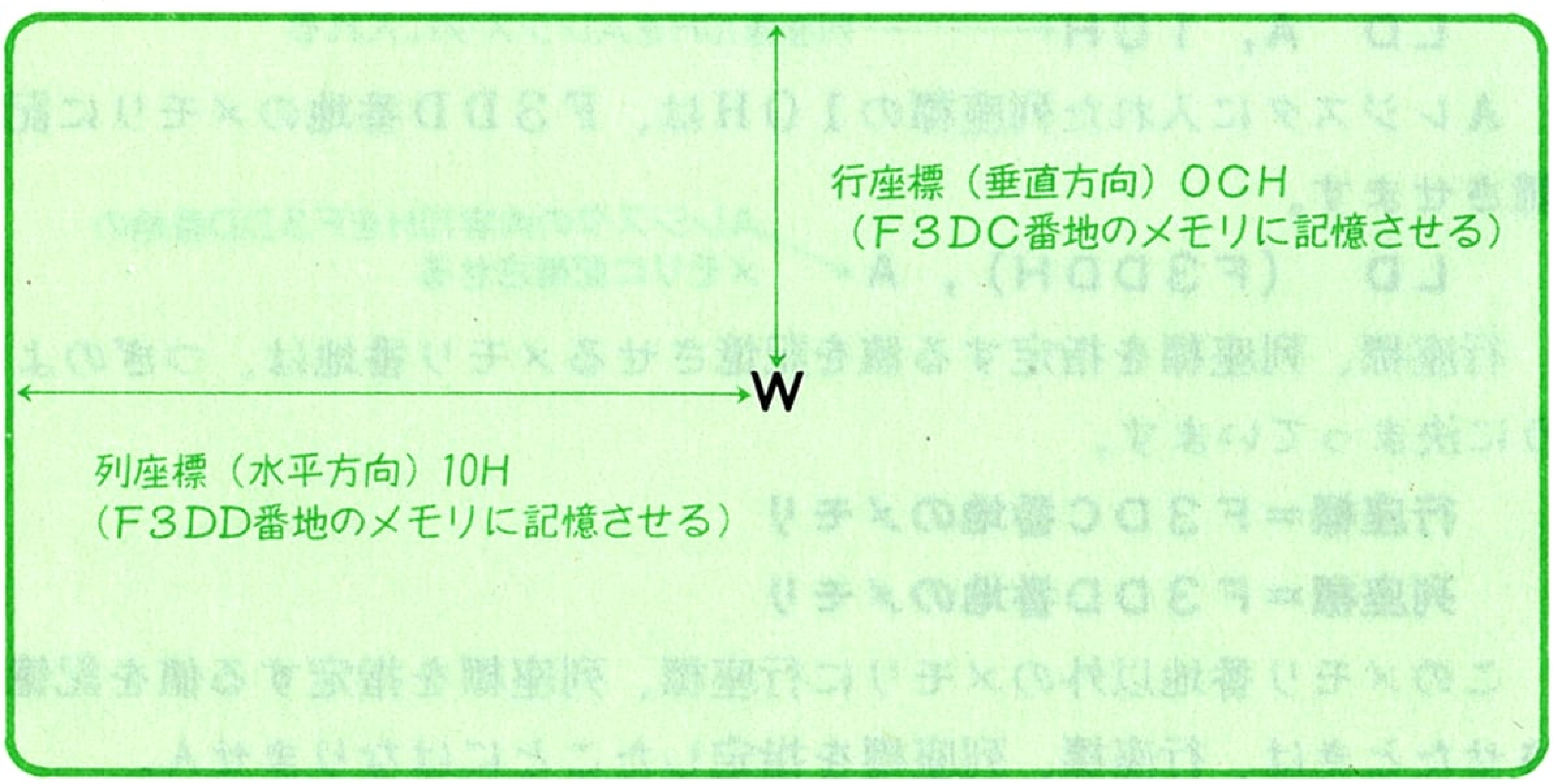
そして、00A2番地のルーチンをコールします。

```
CALL 00A2H
```
←00A2番地のルーチンをコールする

つぎに、これまで示したアセンブリ言語をマシン語になおして、まとめたプログラムを示します。

メモリ番地	マシン語	アセンブリ言語
E000	3E 0C	LD A, 0CH
E002	32 DC F3	LD (F3DCH), A
E005	3E 10	LD A, 10H
E007	32 DD F3	LD (F3DDH), A
E00A	3E 57	LD A, 57H
E00C	CD A2 00	CALL 00A2H
E00F	C9	RET

このマシン語のプログラムをGコマンドで実行させると、つぎの図に示すように行座標（垂直方向）0CH、列座標（水平方向）10Hの位置に、Wが表示されます。



テキスト画面に●を20個描く

ではつぎに、テキスト画面に●を20個つづけて描いてみることにします。この場合は説明の都合上、さきにマシン語とアセンブリ言語のプログラムを示します。

メモリ番地	マシン語	アセンブリ言語
E000	06 14	LD B, 14H
E002	3E 0C	LD A, 0CH
E004	32 DC F3	LD (F3DCH), A
E007	3E 05	LD A, 05H
E009	32 DD F3	LD (F3DDH), A
E00C	3E 85	LD A, 85H
E00E	CD A2 00	CALL 00A2H
E011	10 F9	DJNZ E00CH
E013	C9	RET

●を20個、テキスト画面に表示する場合も、これまでテキスト画面に文字を表示したのとおなじ方法です。ただ、繰り返えす命令が加わるだけにすぎません。

ベーシックでは繰り返えす命令として、FOR～NEXT命令があります。おなじようにマシン語にも、繰り返えす命令としてDJNZ命令があります。

DJNZ命令は、英語のDecriment B, Jump if Non Zeroの略記号です。この意味は、「Bレジスタの内容から1を引く、もし0でなければジャンプする」ということです。これは直訳ですから、なんのことかわからなくて当然です。

つまり、Bレジスタに、DJNZ命令で繰り返えす回数を入れます。そして、DJNZ命令で繰り返えす命令にジャンプするたびに、Bレジスタから1を引いていきます。Bレジスタから1を引いていくと、Bレジスタに入れた値はいつかは0になります。Bレジスタの値が0

になったら、ＤＪＮＺ命令による繰り返えしはおわりとなって、つぎの命令を実行するということです。

ＤＪＮＺ命令＝繰り返えす命令

繰り返えす回数＝Ｂレジスタにセット

Ｂレジスタに１４Ｈを入れる

この場合、20個の●をテキスト画面に表示するわけですから、ＤＪＮＺ命令で20回繰り返えせばよいことになります。20回の20は10進数ですから、16進数になおすと１４Ｈです。したがって、Ｂレジスタに１４Ｈを入れます（プログラムの１行目）。

```
LD  B, 14H ←――― DJNZ命令で繰り返えす回数を入れる
```

つぎは、●を表示する行座標（垂直方向）と列座標（水平方向）の値を決めて、Ｆ３ＤＣ番地とＦ３ＤＤ番地に記憶させます。単にテキスト画面の上に●を20個表示するというときは、列座標、行座標の指定はいりませんから、つぎのプログラムの２行目からプログラムの５行目までの、４つの命令を省略すればよいのです。ここでは、行座標０ＣＨ（10進数で12）の位置で、列座標０５Ｈ（10進数で5）の位置に最初の●を表示させて、20個の●を表示することにしますから、まず、Ａレジスタに０ＣＨを入れます（プログラムの２行目）。

```
LD  A, 0CH ←――― 行座標の0CHをAレジスタに入れる
```

そして、Ａレジスタの内容０ＣＨをＦ３ＤＣ番地のメモリに記憶します（プログラムの３行目）。

```
LD  (F3DCH), A ←――― Aレジスタの内容をF3DC番地の
                     メモリに記憶させる
```

つぎに、列座標の０５ＨをＡレジスタに入れて（プログラム４行目）、

```
LD  A, 05H ←――― 列座標の05HをAレジスタに入れる
```

Ａレジスタの内容０５Ｈを、Ｆ３ＤＤ番地のメモリに記憶させます（プログラムの５行目）。

```
LD  (F3DDH), A ←――― Aレジスタの内容をF3DD番地の
                     メモリに記憶させる
```

つぎに、●のキャラクタコード８５ＨをＡレジスタに入れて（プログラムの６行目）、ＣＡＬＬ命令で、００Ａ２番地のルーチンをコールし

ます（プログラムの7行目）。

```
LD   A, 85H   ←●のキャラクタコード85HをAレジスタに入れる
CALL 00A2H    ←00A2番地のルーチンをコール
```

DJNZ命令で繰り返えす

これで、最初の●が行座標（垂直方向）０ＣＨ、列座標(水平方向）０５Ｈの位置に表示されます。つぎにＤＪＮＺ命令で繰り返えして●を表示させていきますから、ＤＪＮＺ命令に、繰り返えすために戻す最初の命令が記憶されている、メモリ番地を指定します。

DJNZ 〈繰り返えすために戻る最初の命令のメモリ番地〉

さて、ＤＪＮＺ命令で繰り返えす最初の命令はどこかというと、プログラムの１行目は、繰り返えす回数をＢレジスタに入れていますから、ここに戻してしまうと、戻るたびにＢレジスタに１４Ｈが入れられるので、Ｂレジスタが0になることがありません。そのため、永久に繰り返えされてしまいます。プログラムの2行目から5行目は、行座標と列座標を入れているので、ここに戻すと、戻るたびに行座標の０ＣＨと列座標の０５Ｈが入れられてしまうので、●が最初に表示された位置に示されるだけです。もちろん、1行目に戻した場合も、2行目から5行目は実行されるのでおなじです。

とすると、ＤＪＮＺ命令で戻す命令は、プログラムの6行目の、

```
LD   A, 85H   ←DJNZ命令で戻す
```

ということになります。したがって、ＤＪＮＺ命令は、

```
DJNZ E00CH    ←DJNZ命令で戻るメモリ番地を指定
```

となります。なお、アセンブリ言語とマシン語のメモリ番地の書き方が、つぎのように違っています。

```
DJNZ E00CH    ←メモリ番地がE00CH
10   F9       ←メモリ番地がF9
(10 ← DJNZのマシン語)
```

このことについては、226頁の「プログラムカウンタ」で説明します。これでＢレジスタが0になるまでＤＪＮＺ命令で繰り返えされて、●が表示されていきます。

キャラクタコード表

コード		キャラクタ	コード		キャラクタ	コード		キャラクタ	コード		キャラクタ
10進	16進		10進	16進		10進	16進		10進	16進	
32	20		61	3D	=	90	5A	Z	119	77	w
33	21	!	62	3E	>	91	5B	[	120	78	x
34	22	"	63	3F	?	92	5C	¥	121	79	y
35	23	#	64	40	@	93	5D	]	122	7A	z
36	24	$	65	41	A	94	5E	^	123	7B	{
37	25	%	66	42	B	95	5F	_	124	7C	|
38	26	&	67	43	C	96	60	'	125	7D	}
39	27	'	68	44	D	97	61	a	126	7E	~
40	28	(	69	45	E	98	62	b	127	7F	
41	29	)	70	46	F	99	63	c	128	80	♠
42	2A	*	71	47	G	100	64	d	129	81	♥
43	2B	+	72	48	H	101	65	e	130	82	♣
44	2C	,	73	49	I	102	66	f	131	83	♦
45	2D	-	74	4A	J	103	67	g	132	84	○
46	2E	.	75	4B	K	104	68	h	133	85	●
47	2F	/	76	4C	L	105	69	i	134	86	を
48	30	0	77	4D	M	106	6A	j	135	87	ぁ
49	31	1	78	4E	N	107	6B	k	136	88	ぃ
50	32	2	79	4F	O	108	6C	l	137	89	ぅ
51	33	3	80	50	P	109	6D	m	138	8A	ぇ
52	34	4	81	51	Q	110	6E	n	139	8B	ぉ
53	35	5	82	52	R	111	6F	o	140	8C	ゃ
54	36	6	83	53	S	112	70	p	141	8D	ゅ
55	37	7	84	54	T	113	71	q	142	8E	ょ
56	38	8	85	55	U	114	72	r	143	8F	っ
57	39	9	86	56	V	115	73	s	144	90	
58	3A	:	87	57	W	116	74	t	145	91	あ
59	3B	;	88	58	X	117	75	u	146	92	い
60	3C	<	89	59	Y	118	76	v	147	93	う

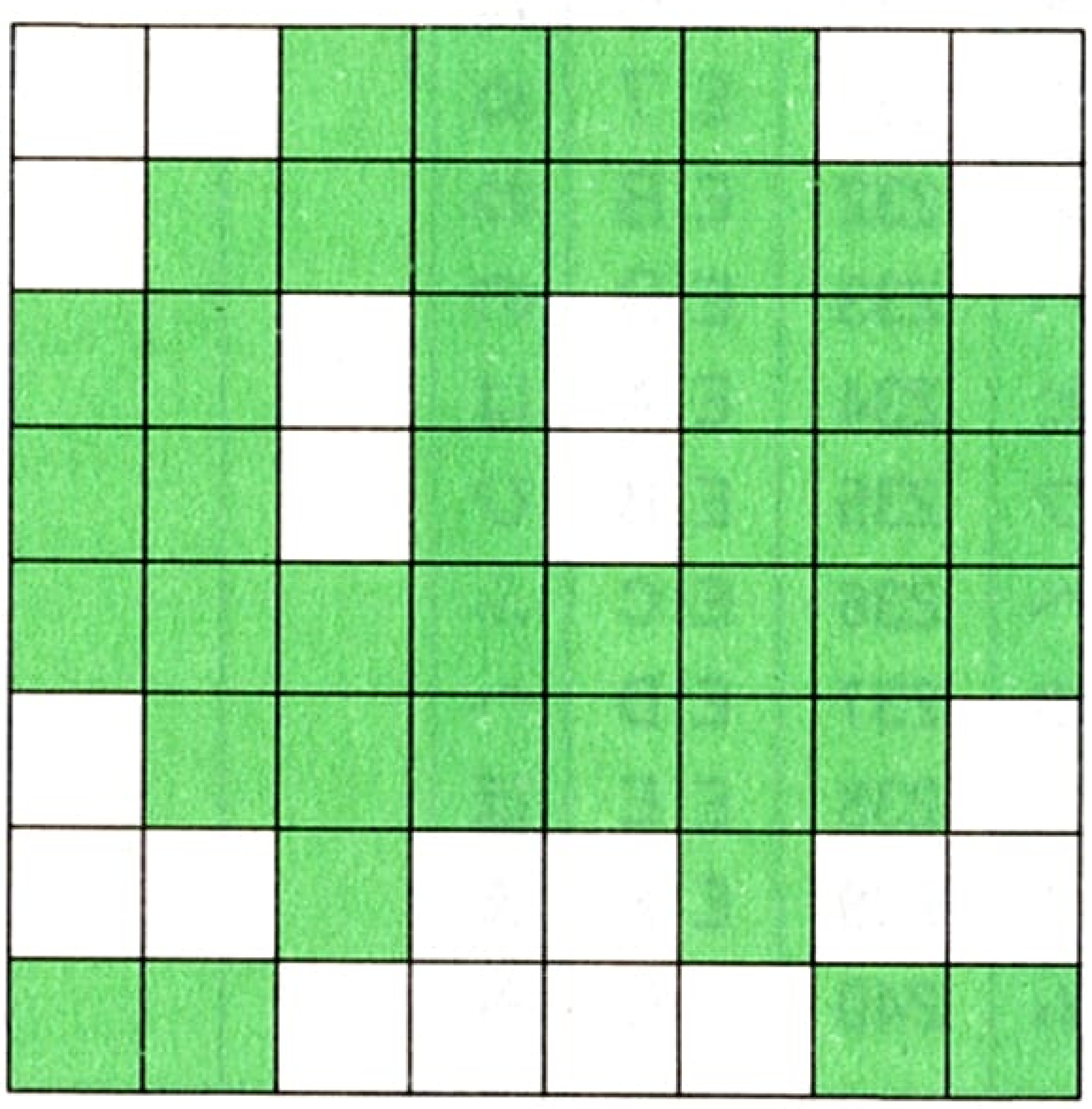
コード		キャラクタ	コード		キャラクタ	コード		キャラクタ	コード		キャラクタ
10進	16進		10進	16進		10進	16進		10進	16進	
148	94	え	179	B3	ウ	210	D2	メ	241	F1	む
149	95	お	180	B4	エ	211	D3	モ	242	F2	め
150	96	か	181	B5	オ	212	D4	ヤ	243	F3	も
151	97	き	182	B6	カ	213	D5	ユ	244	F4	や
152	98	く	183	B7	キ	214	D6	ヨ	245	F5	ゆ
153	99	け	184	B8	ク	215	D7	ラ	246	F6	よ
154	9A	こ	185	B9	ケ	216	D8	リ	247	F7	ら
155	9B	さ	186	BA	コ	217	D9	ル	248	F8	り
156	9C	し	187	BB	サ	218	DA	レ	249	F9	る
157	9D	す	188	BC	シ	219	DB	ロ	250	FA	れ
158	9E	せ	189	BD	ス	220	DC	ワ	251	FB	ろ
159	9F	そ	190	BE	セ	221	DD	ン	252	FC	わ
160	A0		191	BF	ソ	222	DE	゛	253	FD	ん
161	A1	。	192	C0	タ	223	DF	゜	254	FE	
162	A2	「	193	C1	チ	224	E0	た	255	FF	
163	A3	」	194	C2	ツ	225	E1	ち			
164	A4	、	195	C3	テ	226	E2	つ			
165	A5	・	196	C4	ト	227	E3	て			
166	A6	ヲ	197	C5	ナ	228	E4	と			
167	A7	ァ	198	C6	ニ	229	E5	な			
168	A8	ィ	199	C7	ヌ	230	E6	に			
169	A9	ゥ	200	C8	ネ	231	E7	ぬ			
170	AA	ェ	201	C9	ノ	232	E8	ね			
171	AB	ォ	202	CA	ハ	233	E9	の			
172	AC	ャ	203	CB	ヒ	234	EA	は			
173	AD	ュ	204	CC	フ	235	EB	ひ			
174	AE	ョ	205	CD	ヘ	236	EC	ふ			
175	AF	ッ	206	CE	ホ	237	ED	へ			
176	B0	ー	207	CF	マ	238	EE	ほ			
177	B1	ア	208	D0	ミ	239	EF	ま			
178	B2	イ	209	D1	ム	240	F0	み			

スプライト面にスプライトを表示する

スプライトパターンのデータを作る

158頁で、ＢＩＯＳに用意されている００Ａ２番地からの、テキスト画面に文字を表示するルーチンを使って、テキスト画面に文字を表示させました。ここではＢＩＯＳに用意されている００５Ｃ番地からのルーチンを使って、スプライト面にスプライト（キャラクタ）を表示することにします。テキスト画面に文字を表示するのは簡単でしたが、スプライト面にスプライトを表示するのは、少しややこしくなります。なぜなら、スプライト面にスプライトを表示するには、スプライトジェネレータテーブルにスプライトパターンを定義しなければなりませんし、スプライト属性テーブルにスプライトの表示位置や色の指定などを行う必要があるからです。

ではスプライトパターンを、どんな形のものにするかからはじめます。ここではスプライトパターンを、つぎの図に示すものとします。スプライトサイズは8×8ビット、つまり横8ビットの8列です。



➡ 00111100
➡ 01111110
➡ 11010111
➡ 11010111
➡ 11111111
➡ 01111110
➡ 00100100
➡ 11000011

スプライトパターンが決まったら、図の右に示すように、表示する部分（色の部分）を１、表示しない部分（白い部分）を０として、スプライトパターンのデータを書き表わします。ここまでは、ＭＳＸベーシックの場合とおなじです。

１と０で書き表わしたスプライトパターンのデータは、１と０の８個の並びですから、いわゆる８ビットの２進数です。マシン語は16進数で書き表わしますから、この８ビットの２進数を、16進数に書きなおさなければなりません。８ビットの２進数を16進数に書きなおすことについては、98頁で説明していますから参照してください。つぎに16進数で書き表わしたスプライトパターンのデータを示します。

２進数のパターンデータ		16進数のパターンデータ
00111100	➡	3CH
01111110	➡	7EH
11010111	➡	D7H
11010111	➡	D7H
11111111	➡	FFH
01111110	➡	7EH
00100100	➡	24H
11000011	➡	C3H

これで、図に示したスプライトパターンのデータができましたから、つぎは、スプライトジェネレータテーブルに、このデータでスプライトパターンを定義します。スプライトパターンを定義するには、データにメモリ番地をつけておかなければなりません。そこで、つぎのように、このデータにＥ０５０番地からのメモリ番地をつけることにします。

メモリ番地	スプライトパターンのデータ
Ｅ０５０	３Ｃ　７Ｅ　Ｄ７　Ｄ７
Ｅ０５４	ＦＦ　７Ｅ　２４　Ｃ３

お断わりしておきますが、このスプライトパターンのデータにつけたＥ０５０番地からのメモリ番地は、適当につけたわけではありません。スプライト面にスプライトを表示するプログラムはできていて、わかりやすく説明する都合で、このようにしたのです。

スプライトパターンを定義する

さて、スプライトジェネレータテーブルにスプライトパターンを定義します。スプライトジェネレータテーブルにスプライトパターンを定義するには、153頁に示した００５Ｃ番地からのルーチンを使います。００５Ｃ番地からのルーチンは、つぎのようになっています。

メモリ番地	ルーチンの名称	ルーチンの働きと操作
００５Ｃ	ＬＤＩＲＶＭ	メモリからVRAMへブロック転送する ＨＬレジスタに転送もとの先頭番地 ＤＥレジスタに転送さきの先頭番地 ＢＣレジスタに転送するデータの長さ 以上の３つを指定する

これでわかるように、００５Ｃ番地からのルーチンの働きは、ＶＤＰがもっているＶＲＡＭ（表示用画面）へ、データをブロック転送するというものです。

ブロック転送とは、メモリに記憶されている何バイトかのまとまったデータを、そっくりそのまま、ほかの場所に送ることをいいます。

ブロック転送＝何バイトかのデータをまとめて送る

この場合は、16進数２桁で１バイト（８ビット）、それが８個ありますから、８バイトのデータをＶＤＰがもっているＶＲＡＭへ、まとめて送ることになります。

ところで、ただＣＡＬＬ命令で００５Ｃ番地からのルーチンをコールしても、００５Ｃ番地からのルーチンは働いてくれません。なぜなら、うえの表にあるように指定する事柄があるからです。

まず、ＨＬレジスタに転送もとの先頭番地とあります。これは、ＨＬレジスタペアに、データが記憶されている最初のメモリ番地を入れるということです。

ＨＬレジスタペア＝データが記憶されている最初のメモリ番地

この場合は、スプライトパターンのデータには、Ｅ０５０番地からのメモリ番地を割りあてていますから、Ｅ０５０番地をＨＬレジスタペアに入れることになります。

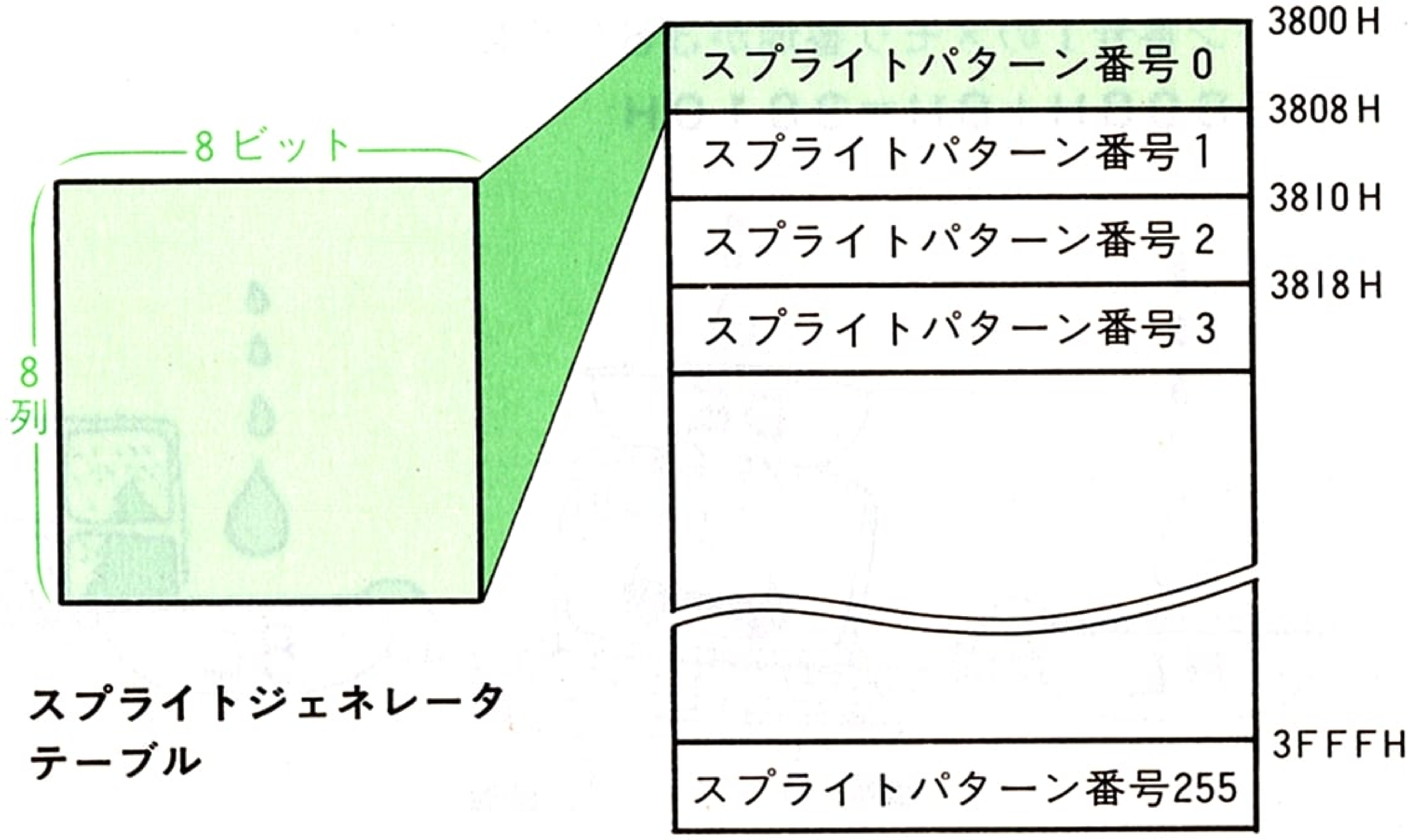
```
LD  HL, E050H
```
← HLレジスタペアにデータの最初のメモリ番地E050Hを入れる

つぎは、ＤＥレジスタに転送さきのメモリ番地とあります。つまりＤＥレジスタペアに、データを送るさきのメモリ番地を入れるということです。

この場合は、スプライトパターンの定義ですから、データを送るさきは、スプライトジェネレータテーブルです。

スプライトジェネレータテーブル

スプライトジェネレータテーブルは、ＶＲＡＭ領域(りょういき)の３８００番地から３ＦＦＦ番地まで割りあてられています。スプライトパターンは８×８ビット（横８ビットの８列＝８バイト）が基本ですから、スプライトジェネレータテーブルのなかは、８バイトのスプライトパター



スプライトジェネレータテーブル

ンが定義できるように、8バイトずつに区切られています。そして、３８００番地のほうから３ＦＦＦ番地のほうに向かって、０、１、２、3…… 255 までのスプライトパターン番号がつけられています。

このようにスプライトジェネレータテーブルは、スプライトパターン番号0から255まであるので、256種類のスプライトパターンを定義することができるのです。

スプライトパターンを、このスプライトジェネレータテーブルに定義するには、スプライトパターン番号の最初のメモリ番地を、ＤＥレジスタペアに入れるわけです。

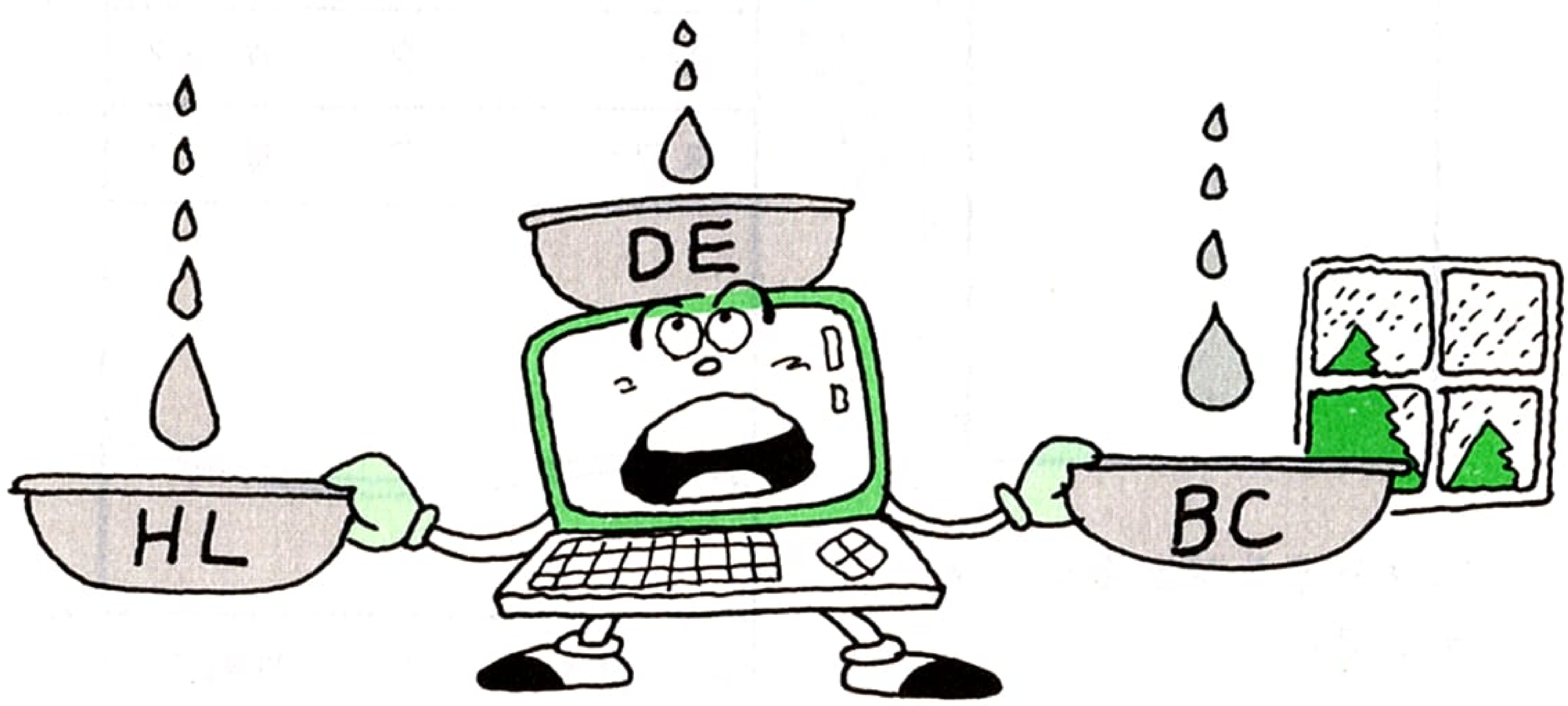
ＤＥレジスタペア＝スプライトパターン番号の最初のメモリ番地

たとえば、スプライトパターンをスプライトパターン番号2に定義したいときは３８１０番地、スプライトパターン番号3に定義したいときは、３８１８番地というようにです。この場合は、スプライトパターンをスプライトパターン番号0に定義するので、DEレジスタペアには、スプライトパターン番号0の最初の３８００番地を入れます。

```
LD  DE, 3800H
```
←スプライトパターン番号0の最初のメモリ番地を入れる

スプライトジェネレータテーブルのメモリ番地を算出するには、最初のメモリ番地に８Ｈをプラスすればよいのです。たとえば、スプライトパターン番号2の最初のメモリ番地を算出するには、スプライトパターン番号1のメモリ番地が３８０８Ｈですから、

3808H+8H=3810H



となりますから、スプライトパターン番号２の最初のメモリ番地は３８１０Ｈとなります。

最後は、ＢＣレジスタペアに転送するデータの長さの指定です。データの長さとは、データのバイト数です。

ＢＣレジスタペア＝データのバイト数

この場合のデータは、16進数２桁が８個ですから、８バイトです。したがって、データの長さは８Ｈとなります。ただし、ＢＣレジスタペアは16ビットレジスタですから、つぎのように入れてはいけません。

```
LD  BC, 8H ←――― 間違い
```

８Ｈのまえに０を３個つけて、16ビットの値にして入れます。

```
LD  BC, 0008H ←――― 0を3個つけて16ビットにして入れる
```

このようにＨＬレジスタペア、ＤＥレジスタペア、ＢＣレジスタペアに指定したら、００５Ｃ番地のルーチンを、ＣＡＬＬ命令でコールします。

```
CALL  005CH ←――― 005CHのルーチンをコール
```

これで、スプライトパターンを定義するアセンブリ言語のプログラムができました。つぎに、これまでのアセンブリ言語をマシン語になおして、まとめて示します。

メモリ番地	マ シ ン 語	アセンブリ言語
E000	21 50 E0	LD HL, E050H
E003	11 00 38	LD DE, 3800H
E006	01 08 00	LD BC, 0008H
E009	CD 5C 00	CALL 005CH
	スプライトパターンのデータ	
E050	3C 7E D7 D7	
E054	FF 7E 24 C3	

これまで、スプライトパターンの定義という言葉を使ってきましたが、スプライトパターンを定義するということは、いいかえれば、ス

プライトジェネレータテーブルに、スプライトパターンのデータを格納するということです。したがって、このプログラムを実行させるとCALL命令で005CHのルーチンがコールされて、E050番地から記憶されているスプライトパターンのデータが、スプライトジェネレータテーブルの3800番地から、つぎのように格納されることになります。そして、このスプライト（キャラクタ）は、スプライトパターン番号0となります。

スプライトパターンのデータ

メモリ番地	データ
E050	3C
E051	7E
E052	D7
E053	D7
E054	FF
E055	7E
E056	24
E057	C3

005CHのルーチンでスプライトジェネレータテーブルの3800番地にブロック転送 ➡

スプライトパターン番号0

メモリ番地	データ
3800	3C
3801	7E
3802	D7
3803	D7
3804	FF
3805	7E
3806	24
3807	C3

さて、これだけではスプライト面にスプライト（キャラクタ）を表示させることができません。スプライト面にスプライトを表示させるには、スプライト属性テーブルに、

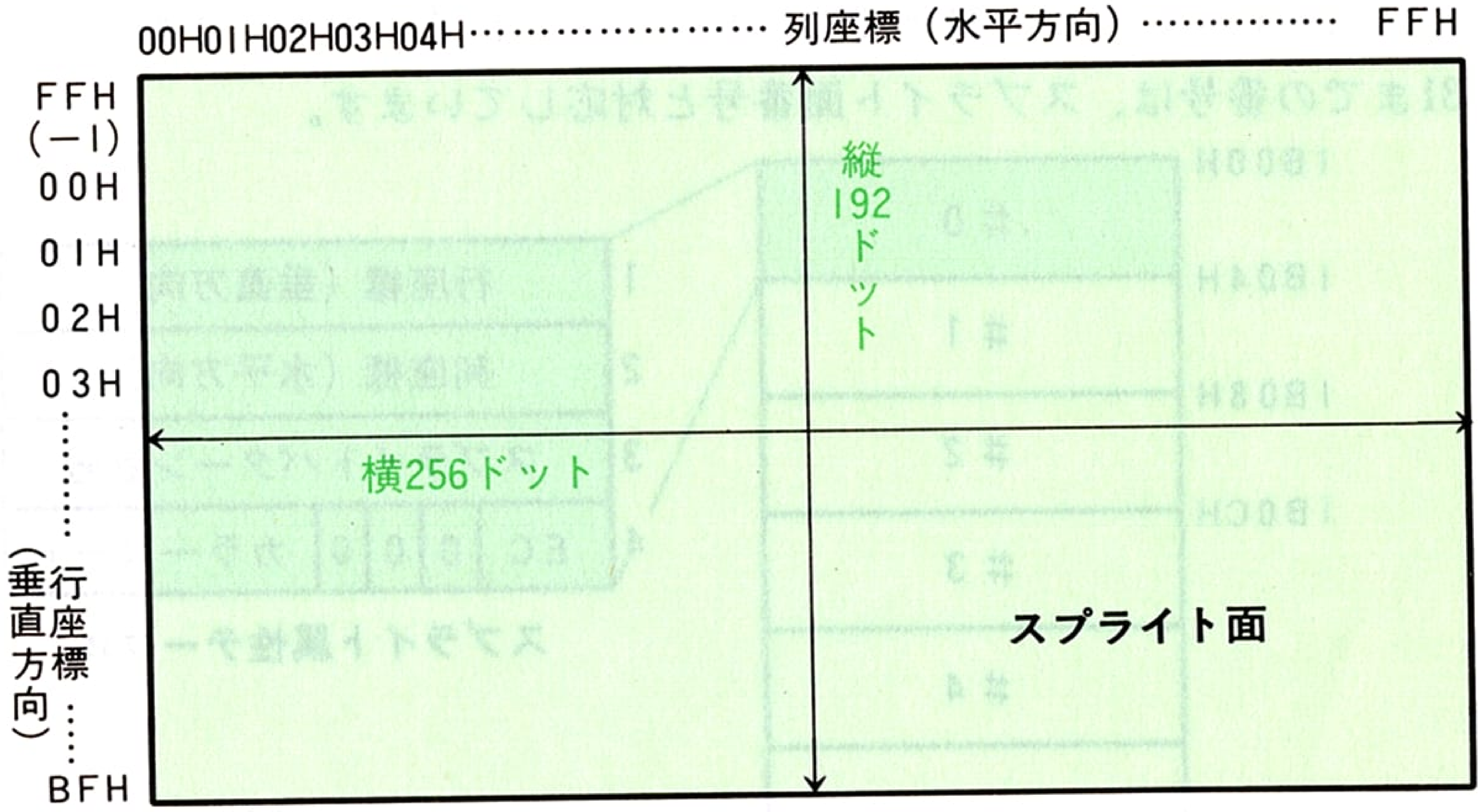
スプライトを表示させる行座標と列座標
データを格納したスプライトパターン番号
表示させるスプライトの色

などを指定しなければならないのです。

スプライト属性テーブルには、スプライトを表示させる行座標と列座標を指定しなければなりませんから、そのまえに、スプライトを表示するスプライト面がどのように構成されているか、みてみることにします。

スプライトを表示するスプライト面の構成

スプライト面は面番号0から31まであって、それぞれのスプライト面にスプライトを1個表示することができるということは、ベーシックの場合とおなじです。ここでみてみたいのは、そういうことではなくて、スプライトを表示させるスプライト面の行座標や列座標がどうつけられているかということです。それぞれのスプライト面には、行座標と列座標は、つぎのようにつけられています。



スプライト面は、横256ドット、縦192ドットになっています。そして、列座標（水平方向）は、００Ｈから０１Ｈ、０２Ｈ……ＦＦＨ(10進数で255)までつけられています。行座標（垂直方向）は、ＦＦＨ(−1）から００Ｈ、０１Ｈ、０２Ｈ……ＢＦＨ（10進数で191）までつけられています。

スプライト面にスプライトを表示する行座標（垂直方向）、列座標（水平方向）の指定は、うえの図に示した、ドット単位につけられた値で行います。

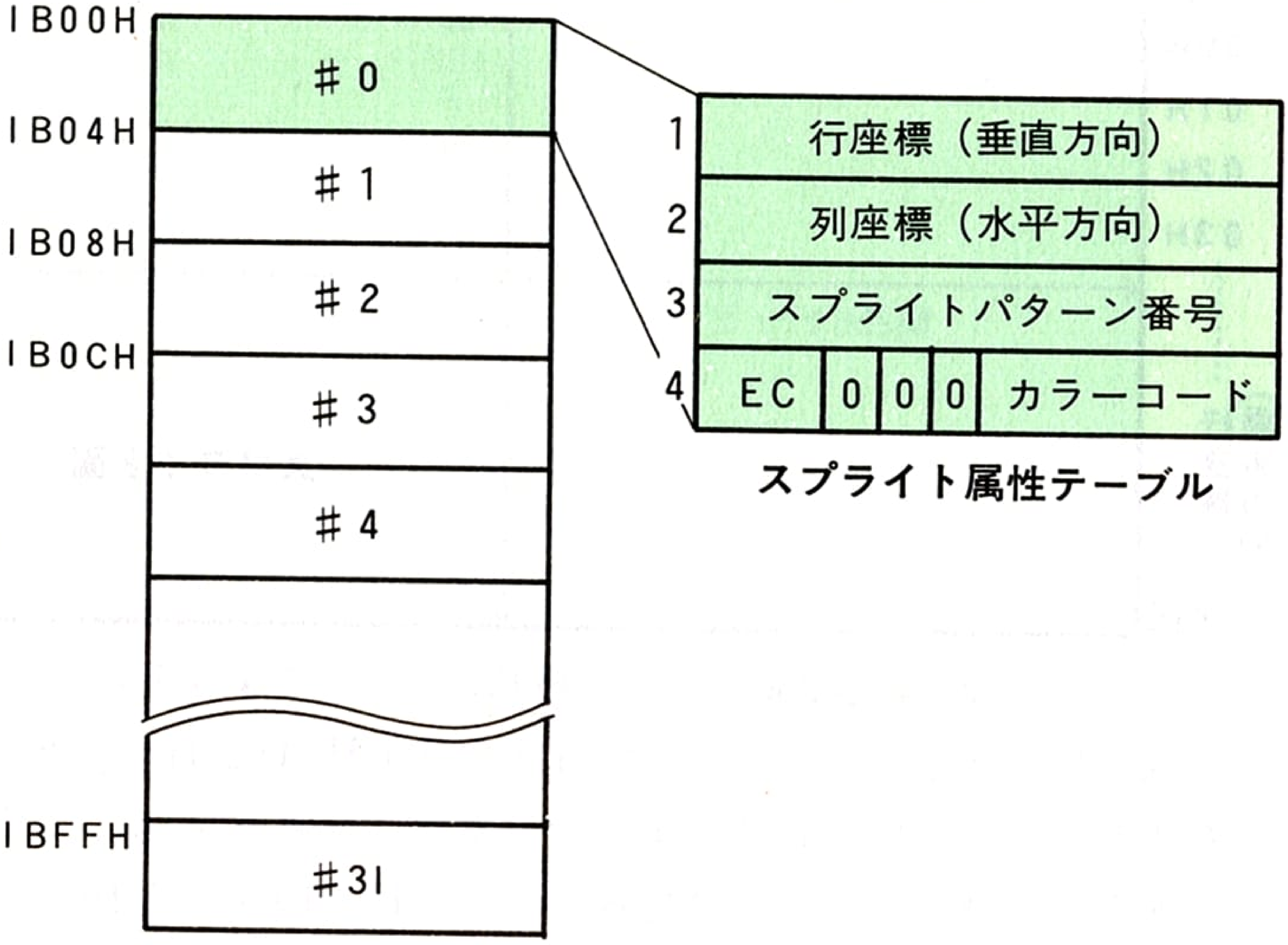
表示位置の指定＝ドット単位につけられた値で行う

スプライトを移動させる場合も、この範囲内となります。176頁でスプライトの移動について説明するのでおぼえておいてください。

表示するスプライトの座標位置、色などを指定

では、スプライトパターン番号０に定義したスプライトを、スプライト面に表示することにします。さきに説明しましたが、スプライトをスプライト面に表示するには、スプライト属性テーブルに必要な事柄を指定しなければなりません。

スプライト属性テーブルは、ＶＲＡＭの１Ｂ００番地から１ＢＦＦ番地までに割りあてられています。そして、そのなかは４バイトずつに区切られて、０から31までの番号がつけられています。この０から31までの番号は、スプライト面番号と対応しています。

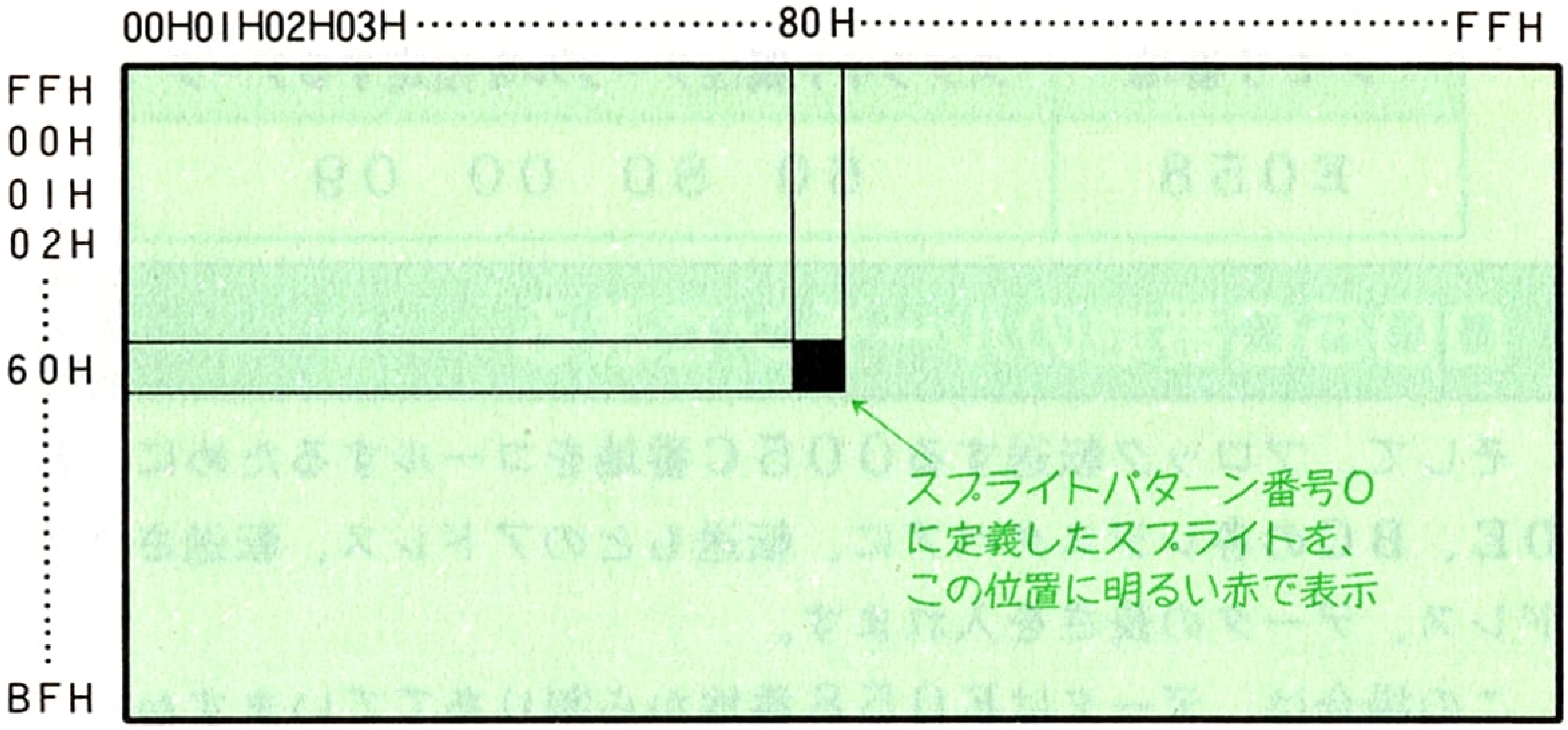


スプライト属性テーブル

スプライト属性テーブルは、うえの図に示すようになっていますが、指定する事柄は、行座標、列座標、スプライトパターン番号、それにカラーコードの４つです。

スプライト属性テーブルに指定する場合も、００５Ｃ番地のブロック転送のルーチンを使って行いますから、まず、データを作ります。

この場合は、行座標６０Ｈ、列座標８０Ｈにスプライトを表示することにします。



1　行座標＝６０Ｈ　　　　**2　列座標＝８０Ｈ**

スプライトパターン番号は0です。0という1桁のときは、16進数2桁にして００Ｈとします。

3　スプライトパターン番号＝００Ｈ

カラーコードは、下に示したカラーコード表をみて決めます。この場合は、スプライトを明るい赤で表示するので、０９Ｈとします。

4　カラーコード＝０９Ｈ

これでスプライト属性テーブルを指定するデータができました。この６０Ｈ、８０Ｈ、００Ｈ、０９Ｈを、スプライトパターンのデータのあとにおきます。スプライトパターンのデータは、Ｅ０５０番地からＥ０５７番地まででしたから、Ｅ０５８番地からとなります。

カラーコード	色	カラーコード	色
００Ｈ	透明	０８Ｈ	赤
０１Ｈ	黒	０９Ｈ	明るい赤
０２Ｈ	緑	０ＡＨ	黄色
０３Ｈ	明るい緑	０ＢＨ	明るい黄色
０４Ｈ	暗い青	０ＣＨ	暗い緑
０５Ｈ	明るい青	０ＤＨ	紫
０６Ｈ	暗い赤	０ＥＨ	灰色
０７Ｈ	水色	０ＦＨ	白

メモリ番地	スプライト属性テーブルを指定するデータ
E058	60　80　00　09

005C番地のルーチンで送る

そして、ブロック転送する005C番地をコールするために、HL、DE、BCの各レジスタペアに、転送もとのアドレス、転送さきのアドレス、データの長さを入れます。

この場合は、データはE058番地から割りあてていますから、HLレジスタペアには、転送もとのアドレスE058番地を入れます。

```
LD  HL, E058H  ←データの最初のメモリ番地を入れる
```

つぎは、DEレジスタペアに転送さきのメモリ番地を入れます。この場合は、スプライト面番号0（♯0）のスプライト面にスプライトを表示するので、スプライト面番号0のスプライト属性テーブルにデータを送ります。したがってDEレジスタペアに、172頁の図に示した最初のメモリ番地1B00番地を入れます。

```
LD  DE, 1B00H  ←1B00Hを入れる
```

スプライト面番号1（♯1）のスプライト面にスプライトを表示するときは、DEレジスタペアに、172頁の図に示した2番目のメモリ番地1B04Hを入れて、1B04番地にデータを送ります。

スプライト面番号2（♯2）のスプライト面にスプライトを表示するときは、DEレジスタペアに1B08Hを入れて、1B08番地にデータを送ります。

各スプライト面番号の、スプライト属性テーブルのメモリ番地の算出の仕方は、図に示したメモリ番地に4Hをプラスすればよいのです。

たとえばスプライト面番号3（♯3）の、スプライト属性テーブルのメモリ番地を算出するには、スプライト面番号2（♯2）の、スプライト属性テーブルの1B08番地に4Hをプラスします。

1B08H+4H=1B0CH

で、スプライト面番号3（♯3）のスプライト属性テーブルのメモ

り番地が求められます。

最後は、BCレジスタペアにデータの長さを入れます。この場合のデータの長さは、16進数2桁が4個ですから、BCレジスタには、04Hを16ビットにして0004Hを入れます。

```
LD  BC, 0004H←――――データの長さを入れる
```

そして、005C番地のルーチンをコールします。

```
CALL  005CH←――――ブロック転送のルーチンをコールする
```

これで、スプライトパターン番号0に定義したスプライトが、スプライト面番号0のスプライト面の、173頁の図に示した位置に表示されます。つぎに、全部をまとめたプログラムを示します。なおこのプログラムの実行結果は、53頁の写真で示していますから参照してください。

メモリ番地	マシン語	アセンブリ言語
		スプライトパターンの定義
E000	21 50 E0	LD HL, E050H
E003	11 00 38	LD DE, 3800H
E006	01 08 00	LD BC, 0008H
E009	CD 5C 00	CALL 005CH
		スプライト属性テーブルの指定
E00C	21 58 E0	LD HL, E058H
E00F	11 00 1B	LD DE, 1B00H
E012	01 04 00	LD BC, 0004H
E015	CD 5C 00	CALL 005CH
E018	C9	RET
E050	3E 7E D7	“データ” 3C 7E D7
E053	D7 FF 7E	D7 FF 7E 24 C3
E056	24 C3 60	“データ” 60 80 00
E059	80 00 09	09

スプライトを右に移動させる

スプライトを左から右へ移動させるには

164頁で、スプライト面に、スプライトを明るい赤で表示しました。ここでは、そのスプライトを右に移動させることにします。スプライトパターンの定義、スプライト属性テーブルの指定は、164頁で説明したこととおなじですから、さきにプログラムを示して、そのあと説明することにします。

メモリ番地	マシン語	アセンブリ言語
E000	21 50 E0	LD HL, E050H
E003	11 00 38	LD DE, 3800H
E006	01 08 00	LD BC, 0008H
E009	CD 5C 00	CALL 005CH
E00C	06 F0	LD B, F0H
E00E	C5	PUSH BC
E00F	21 58 E0	LD HL, E058H
E012	11 00 1B	LD DE, 1B00H
E015	01 04 00	LD BC, 0004H
E018	CD 5C 00	CALL 005CH
E01B	3A 59 E0	LD A, (E059H)
E01E	3C	INC A
E01F	32 59 E0	LD (E059H), A
E022	3E FF	LD A, FFH
E024	06 FF	LD B, FFH

E026	10 FE	DJNZ E026H
E028	3D	DEC A
E029	20 F9	JR NZ E024H
E02B	C1	POP BC
E02C	10 E0	DJNZ E00EH
E02E	C9	RET

E050	3C 7E D7	"データ" 3C 7E D7
E053	D7 FF 7E	D7 FF 7E 24 C3
E056	24 C3 60	"データ" 60 00
E059	00 00 09	00 09

スプライトパターンの定義

スプライトをスプライト面に表示するには、まず、スプライトパターンのデータをスプライトジェネレータテーブルに送って、スプライトパターンの定義をしなければなりません。スプライトジェネレータテーブルにスプライトパターンのデータを定義するには、005C番地のVRAMにブロック転送するルーチンを使います。005C番地のルーチンを使うには、HLレジスタペアに、転送するデータが記憶されているメモリ番地を入れ、DEレジスタペアに、転送するさきのスプライトジェネレータテーブルのメモリ番地を入れます。またBCレジスタペアには、ブロック転送するデータのバイト数を入れます。詳しいことは、168頁を参照してください。E000番地からE009番地までが、スプライトパターンを定義する命令です。

```
LD   HL, E050H ←転送するデータが記憶されている最初の
                メモリ番地を入れる
LD   DE, 3800H ←転送さきのスプライトジェネレータテー
                ブルのメモリ番地を入れる
LD   BC, 0008H ←転送するデータのバイト数を入れる
CALL  005CH ←――ブロック転送するルーチンをコールする
```

1行目で、HLレジスタペアにE050番地を入れているのは、スプライトパターンのデータが、E050番地から記憶されているからです。

2行目で、DEレジスタペアに3800番地を入れているのは、この場合、スプライトパターンを、スプライトパターン番号0に定義するためです。スプライトパターン番号0のスプライトジェネレータテーブルのメモリ番地は3800番地なので、3800Hを入れているわけです。

3行目で、BCレジスタペアに0008Hを入れているのは、スプライトジェネレータテーブルにブロック転送する、データのバイト数が8バイトだからです。BCレジスタペアは、16ビットレジスタです。16ビットのレジスタには、16ビットにして入れなければならないので、0008Hにしているわけです。

そして、4行目のCALL命令で005C番地のルーチンをコールすると、E050番地から記憶されている8バイトのスプライトパターンデータが、3800番地のスプライトジェネレータテーブルに送られて、スプライトパターン番号0に定義されます。

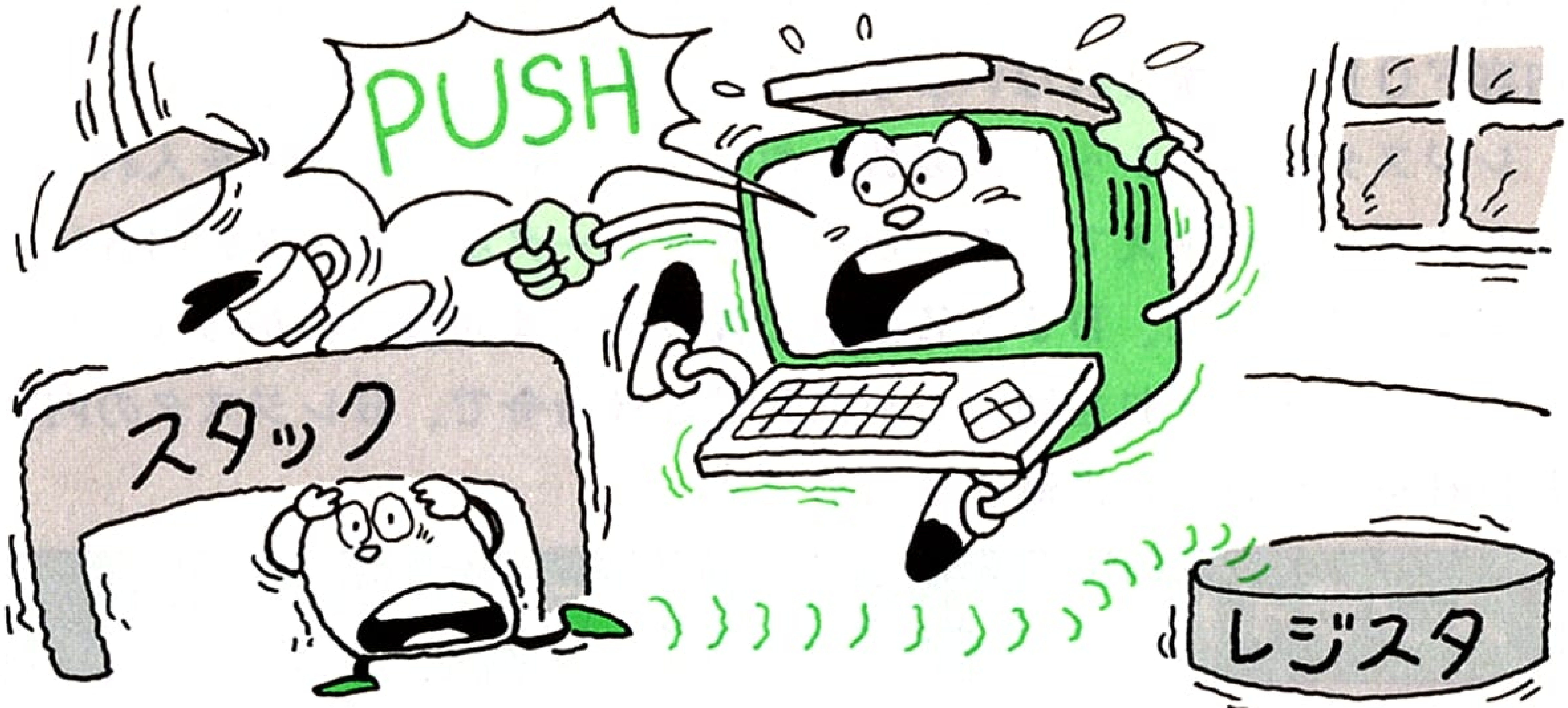
Bレジスタに繰り返えす回数を入れる

MSXベーシックでキャラクタを移動させるとき、キャラクタの表示をFOR～NEXT命令で繰り返えします。マシン語の場合も、繰り返えしの命令DJNZ命令で、スプライトの表示を繰り返えして移動させていきます。DJNZ命令については、159頁で説明していますから、参照してください。

DJNZ命令で繰り返えすときは、Bレジスタに、DJNZ命令で何回繰り返えすのか、繰り返えす回数を入れることになっています。したがって、4行目のCALL　005CHのつぎで、Bレジスタに繰り返えす回数を入れます。

```
LD  B, F0H ←――DJNZ命令で繰り返えす回数を入れる
```

F0Hは、10進数で240です。240をBレジスタに入れているのは、



240回繰り返えして、スプライトを1ドットずつ右にずらしていくためです。

つぎに、Bレジスタに入れたF0Hを、PUSH（プッシュ）命令でスタックに退避（たいひ）させます。退避させるとは、いったんレジスタに記憶されているデータを、スタックに記憶させることです。スタックについては244頁で説明します。ここでは「データの記憶場所」とおぼえておいてください。

PUSH　BC←――Bレジスタの内容F0Hをスタックに退避させる

PUSH命令のPUSHは、英語のPUSH（プッシュ）です。

PUSH命令は、レジスタに記憶されている内容をスタックに退避させるときに使います。

PUSH命令=レジスタの内容をスタックに退避させる

PUSH命令のあとには、かならずレジスタペアをおきます。

PUSH　<レジスタペアを置く>

この場合は、Bレジスタの内容F0Hをスタックに退避させるわけですが、つぎのようにすると、

PUSH　B←――間違い

間違いです。まえのように、BのあとにCをつけて、BCレジスタペアにしなければなりません。

ところで、PUSH命令で、Bレジスタの内容F0Hをスタックに

退避させる理由は、このあとでＢレジスタを使うので、Ｂレジスタの内容Ｆ０Ｈが破壊されないようにするためです。

レジスタはベーシックの変数とおなじで、新たにデータを入れると、まえのデータが消えて、あとから入れたデータがレジスタの内容になります。それでは、ＤＪＮＺ命令で繰り返えす回数がわからなくなってしまいます。それを防ぐためにＰＵＳＨ命令で、Ｂレジスタの内容Ｆ０Ｈをスタックに退避させるのです。

スプライト属性テーブルを指定する

さて、ＰＵＳＨ命令でスタックにＢレジスタの内容Ｆ０Ｈを退避させたら、スプライトパターン番号0に定義したスプライトを、スプライト面に表示させるためにスプライト属性テーブルを指定します。スプライト属性テーブルの指定については、172頁を参照してください。

この場合も、ＶＲＡＭにブロック転送する００５Ｃ番地のルーチンを使いますから、ＨＬレジスタに、データが記憶されている最初のメモリ番地、ＤＥレジスタに、転送するさきのスプライト属性テーブルのメモリ番地、ＢＣレジスタに、転送するバイト数を入れます。

Ｅ００Ｆ番地からＥ０１８番地までの命令が、スプライト属性テーブルの指定です。

```
LD  HL, E058H ←データが記憶されている最初のメモリ番地を入れる
LD  DE, 1B00H ←スプライト属性テーブルのメモリ番地を入れる
LD  BC, 0004H ←転送するデータのバイト数を入れる
CALL  005CH ←ブロック転送するルーチンをコールする
```

１行目でＨＬレジスタペアにＥ０５８Ｈを入れているのは、スプライト属性テーブルの指定をするデータが、Ｅ０５８番地から記憶されているためです。

２行目で、ＤＥレジスタペアに１Ｂ００Ｈを入れているのは、スプライト面番号0のスプライト面にスプライトを表示するためです。この場合のスプライト属性テーブルのメモリ番地は１Ｂ００番地なので、

ＤＥレジスタペアに１Ｂ００Ｈを入れているのです。

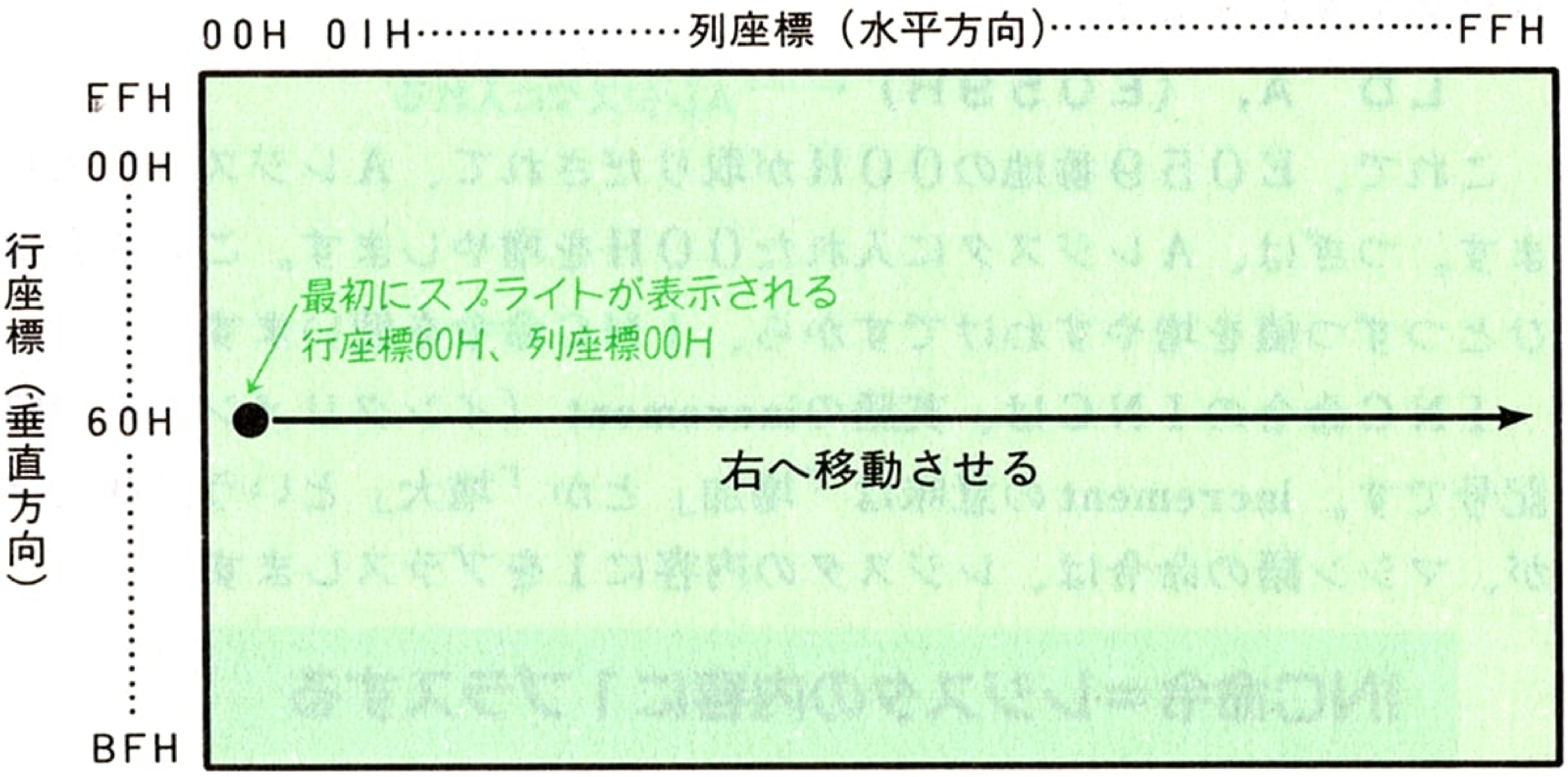
３行目で、ＢＣレジスタペアに０００４Ｈを入れているのは、スプライト属性テーブルを指定するデータが、４バイトだからです。

４行目のＣＡＬＬ命令で００５Ｃ番地のルーチンをコールすると、Ｅ０５８番地から記憶されている４バイトのデータが１Ｂ００番地のスプライト属性テーブルに送られて、指定した位置に、指定した色で、スプライトパターン番号０に定義したスプライトが表示されます。

Ｅ０５０番地からのスプライトパターンのデータは、164頁の図に示したのとおなじですが、Ｅ０５８番地からのデータは、

行座標６０Ｈ　　**列座標００Ｈ**
パターン番号００Ｈ　　**カラーコード０９Ｈ**

ですから、最初に表示されるスプライトは、つぎの図に示す位置に明るい赤で表示されます。



ここでは、最初にスプライトが表示される図の位置、つまり、行座標６０Ｈ、列座標００Ｈの位置から、右に向けてスプライトを移動させていきます。

スプライトを右に移動させていくには、列座標（水平方向）の値をひとつひとつ変化させて、スプライトの表示位置を変えていけばよいわけです。この場合、列座標の値は右に向かって増えていきますから、列座標の値をひとつひとつ増やしていくことになります。

列座標の値を増やすには

列座標の値を増やすには、スプライト属性テーブルの列座標を指定するデータを増やします。スプライト属性テーブルを指定するデータを増やす方法は、E059番地に列座標を指定するデータ00Hが記憶されていますから、つぎのようにすればよいのです。

E058　60（行座標のデータ）
E059　00（列座標のデータ）

Aレジスタに E059番地の00Hを取りだしてくる

↓

Aレジスタの内容00Hに1をプラスする

E059番地の00Hを取りだしてきて、Aレジスタに入れるには、LD命令のあとにAレジスタをおき、そのあとにカッコで括ってE059番地をおきます。

```
LD　A,（E059H）
```
← E059番地の内容を取り出してAレジスタに入れる

これで、E059番地の00Hが取りだされて、Aレジスタに入ります。つぎは、Aレジスタに入れた00Hを増やします。この場合はひとつずつ値を増やすわけですから、**INC命令**を使います。

INC命令のINCは、英語のincrement（インクリメント）の略記号です。incrementの意味は「増加」とか「増大」という意味ですが、マシン語の命令は、レジスタの内容に1をプラスします。

INC命令＝レジスタの内容に1プラスする

この場合は、Aレジスタの内容に1をプラスするわけですから、

```
INC　A
```
← Aレジスタの内容に1プラスする

です。このINC　A命令を繰り返すと、Aレジスタの内容は、

1回目	00H＋1H＝01H	4回目	03H＋1H＝04H
2回目	01H＋1H＝02H	5回目	04H＋1H＝05H
3回目	02H＋1H＝03H	6回目	05H＋1H＝06H

というようにひとつずつ増えていきます。

どうやってスプライトを移動させるのか

さて、列座標の値をひとつずつ増やすことができました。では、どうやってスプライトを移動させるのでしょうか。つぎのようにするとスプライトを移動させることができます。

1プラスしたAレジスタの内容を、もとのE059番地に戻す

↓

スプライト属性テーブルに4バイトのデータを送る

1プラスしたAレジスタの内容をE059番地に戻すには、LD命令のあとにカッコで括ってE059番地をおき、そのつぎにAレジスタをおきます。

```
LD  (E059H), A
```
←Aレジスタの内容をE059番地に戻す

これでAレジスタの内容は、E059番地に記憶されますから、スプライト属性テーブルを指定するデータはつぎのようになります。

E058	60(行座標)
E059	00(列座標)
E05A	00(パターン番号)
E05B	09(カラーコード)

→

E058	60(行座標)
E059	01(列座標)
E05A	00(パターン番号)
E05B	09(カラーコード)

右側のデータをスプライト属性テーブルに送ると、列座標を指定するデータが01になっていますから、スプライトは、1ドット右側に表示されることになります。このことを繰り返すと、列座標を指定するデータは、さきに説明したように、INC　A命令でひとつずつ増えていきますから、スプライトは1ドットずつ右に表示されていきます。これがスプライトを右に移動させる方法です。

Aレジスタに、列座標を指定する値を取りだしてきて1プラスする、1プラスしたデータをもとのE059番地に戻す、スプライト属性テーブルにデータを送る、このことを繰り返すのはDJNZ命令ですが、DJNZ命令で繰り返すためには、PUSH命令で、スタック

に退避させたBレジスタの内容を取りだしてこなければなりません。

なぜなら、DJNZ命令で繰り返えす最初の命令に戻るとき、Bレジスタの内容から1を引いて戻るからです。スタックに退避させたままでは1を引くことができません。そこで、Bレジスタにデータを取りだしてくるのです。

スタックからデータを取りだしてくる命令は、**POP（ポップ）命令**です。POPは英語のPOP（ポップ）です。英語のPOPは「ひょいと出る（入る）」というような意味です。

POP命令＝スタックからデータを取りだす

POP命令のあとには、データを入れるレジスタペアを置きます。この場合も、PUSH命令のときのように、レジスタペアでなければなりません。

POP　〈レジスタペアを置く〉

ここでは、PUSH命令でスタックに退避させたデータを、Bレジスタに取りだしてくるわけですから、

POP　BC←――BCレジスタにスタックからデータを取り出す

となります。POP命令でBレジスタにデータを取りだしてきたら、つぎに、DJNZ命令で繰り返えす最初の命令に戻るとき、1を引くことができますから、つぎにDJNZ命令をおきます。



DJNZ命令で繰り返えす最初の命令は

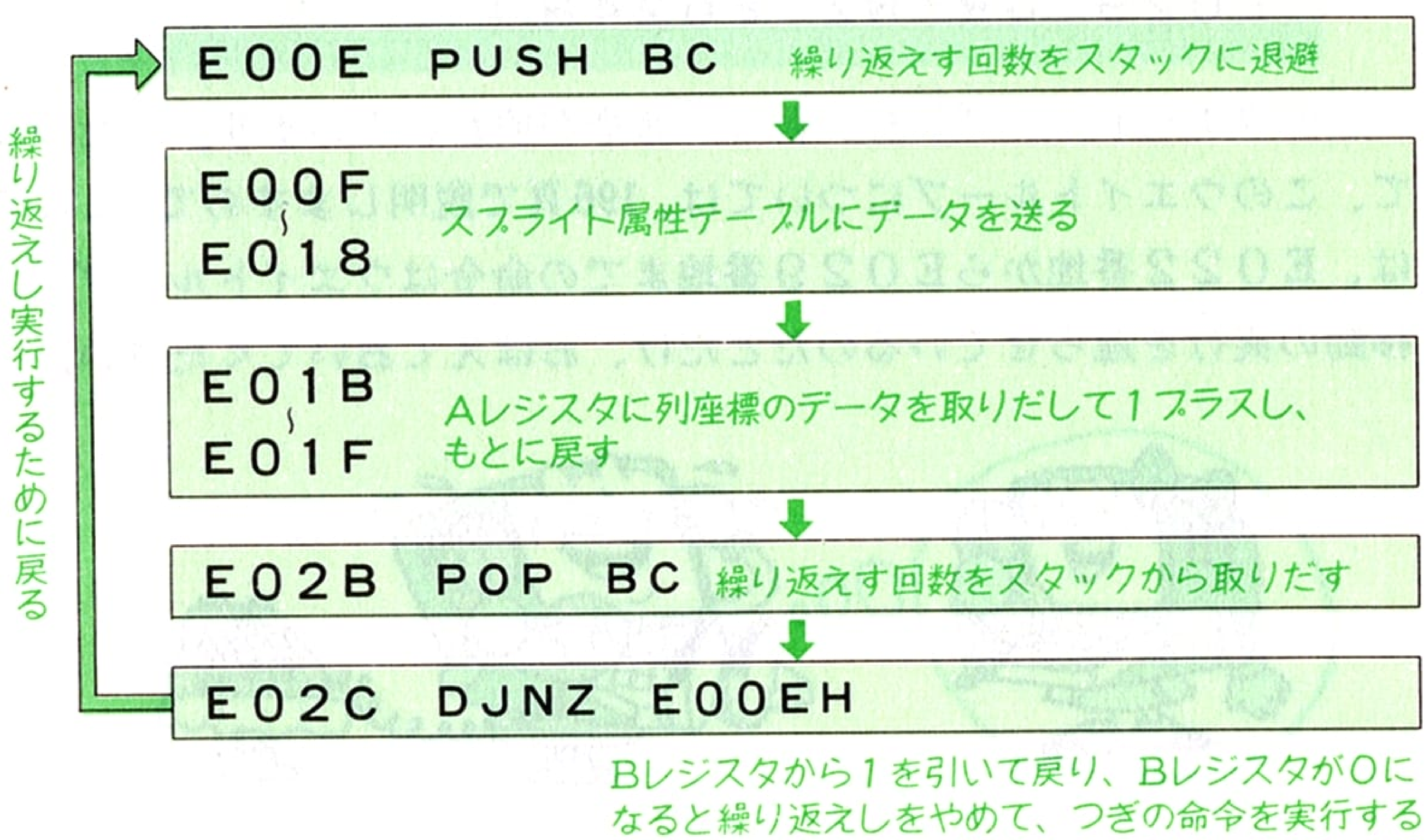
ＤＪＮＺ命令には、ＤＪＮＺ命令で繰り返えす最初の命令があるメモリ番地を指定しなければなりません。繰り返えし働かせる命令としては、まえに説明したＥ００Ｆ番地からＥ０１８番地までのスプライト属性テーブルにデータを送る命令、Ｅ０１Ｂ番地からＥ０１Ｆ番地までの、Ａレジスタに列座標を指定するデータを取りだしてきて１プラスして、またデータを戻す命令があります。

繰り返えし働かせる命令は、もうひとつあります。Ｅ００Ｅ番地のＢレジスタに入れた繰り返えしの回数を、スタックに退避させるＰＵＳＨ　ＢＣ命令です。なぜなら、スプライト属性テーブルにデータを送るとき、ＬＤ　ＢＣ，０００４Ｈ命令で、Ｂレジスタに０００４Ｈを新たに入れるからです。この命令も繰り返えされますから、ＰＵＳＨ　ＢＣ命令も繰り返えさなければならないのです。

したがって、繰り返えし働かせる最初の命令は、Ｅ００Ｅ番地のＰＵＳＨ　ＢＣ命令ですから、ＤＪＮＺ命令には、Ｅ００Ｅ番地を指定することになります。

```
DJNZ  E00EH
```
←DJNZ命令で繰り返えす最初のメモリ番地E00Eを指定

つぎに図で示します。



図に示したように、ＤＪＮＺ命令で繰り返えして、スプライトを右へ移動させていきます。ＤＪＮＺ命令で繰り返えし命令を処理するためにＰＵＳＨ　ＢＣ命令に戻るとき、Ｂレジスタの内容から１を引いていきますから、Ｂレジスタの内容はそのうち０になります。Ｂレジスタの内容が０になると、ＤＪＮＺ命令による繰り返えしはおわりとなって、つぎの命令が実行されます。

このプログラムでは、ＤＪＮＺ命令のつぎはＲＥＴ命令ですから、マシン語モニタに戻ります。

ウエイトループ

これまで説明した命令だけで実行させると、実行速度が非常に速いので、スプライトが移動しているようにはみえません。

そこで移動を遅らせるためにウエイトループを入れています。つぎに、ウエイトループのアセンブリ言語だけを取りだして示します。

```
E022  LD  A, FFH     ┐
E024  LD  B, FFH     │
E026  DJNZ  E026H    ├ ウエイトループ
E028  DEC  A         │
E029  JR  NZ  E024H  ┘
```

このウエイトループにも、新しい命令が使われています。したがって、このウエイトループについては、196頁で説明しますので、ここでは、Ｅ０２２番地からＥ０２９番地までの命令はウエイトループで、移動の実行を遅らせているのだとだけ、おぼえておいてください。



スプライトを斜めに移動する

画面の左上から右下に向けて移動させるには

176頁で、スプライトを画面の左から右へ移動させました。ここではスプライトを画面の左上から右下に向けて斜めに移動させることにします。さきにプログラムを示します。説明は、そのあとで行います。

メモリ番地	マシン語		アセンブリ言語
E000	21 50 E0		LD HL, E050H
E003	11 00 38		LD DE, 3800H
E006	01 08 00		LD BC, 0008H
E009	CD 5C 00		CALL 005CH
E00C	06 A0		LD B, A0H
E00E	C5		PUSH BC
E00F	21 58 E0		LD HL, E058H
E012	11 00 1B		LD DE, 1B00H
E015	01 04 00		LD BC, 0004H
E018	CD 5C 00		CALL 005CH
E01B	3A 58 E0		LD A, (E058H)
E01E	3C		INC A
E01F	32 58 E0		LD (E058H), A
E022	3A 59 E0		LD A, (E059H)
E025	3C		INC A
E026	32 59 E0		LD (E059H), A
E029	3E FF		LD A, FFH
E02B	06 FF		LD B, FFH

E02D	10 FE	DJNZ E02DH
E02F	3D	DEC A
E030	20 F9	JR NZ E02BH
E032	C1	POP BC
E033	10 D9	DJNZ E00EH
E035	C9	RET
E050	3C 7E D7	“データ” 3C 7E
E053	D7 FF 7E	D7 D7 FF 7E
E056	24 C3 00	24 C3
E059	30 00 0F	“データ” 00 30 00 0F

スプライトパターンの定義

プログラムのE000番地からE009番地までの命令は、スプライトパターンのデータを、スプライトジェネレータテーブルに定義する命令です。

```
E000 LD HL, E050H ←HLレジスタペアにデータが記憶されている最初のメモリ番地を入れる
E003 LD DE, 3800H ←スプライトジェネレータテーブルのメモリ番地を入れる
E006 LD BC, 0008H ←スプライトジェネレータテーブルに転送するデータのバイト数を入れる
E009 CALL 005CH ←VDPのVRAMへデータをブロック転送するルーチンをコールする
```

これらの命令は177頁で説明しましたから、簡単に説明します。

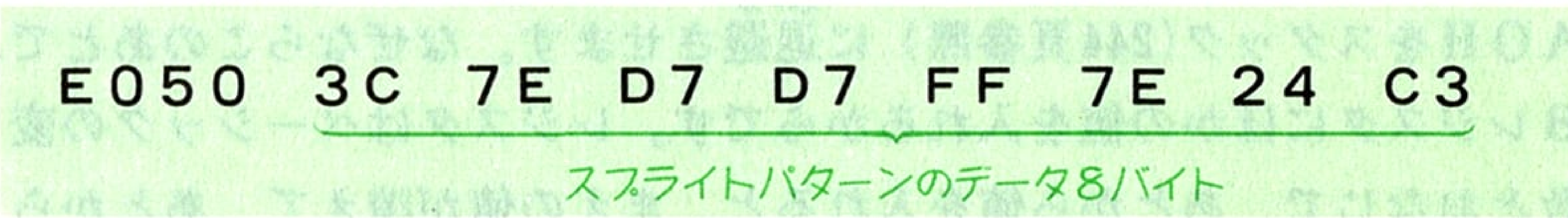
スプライトパターンを、スプライトジェネレータテーブルに定義する場合、VDPのVRAMにデータをブロック転送する005C番地

のルーチンを使います。

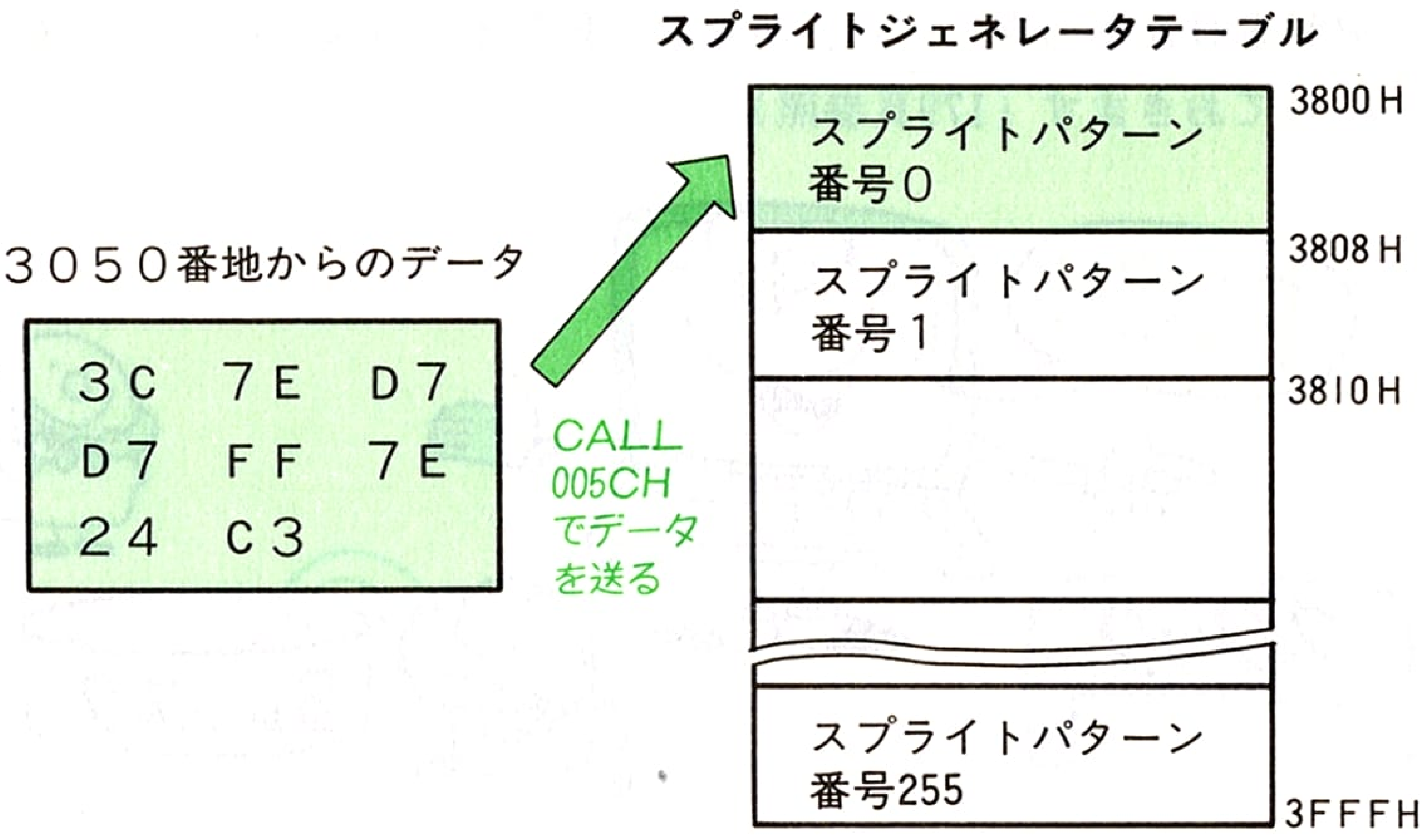
このルーチンを使うときは、HLレジスタペアに、データが記憶されているメモリ番地を入れ、DEレジスタペアには、転送するさきのスプライトジェネレータテーブルのメモリ番地を入れ、BCレジスタペアには、転送するデータのバイト数を入れます。

このプログラムのスプライトパターンは、164頁で図に示したものです。スプライトパターンのデータは、E050番地が割りあてられていますから、HLレジスタペアには、E050番地を入れています。

スプライトパターンは、スプライトパターン番号0に定義するので、DEレジスタペアに、スプライトジェネレータテーブルの3800番地を入れています。また、転送するデータは8バイトですから、BCレジスタペアには0008Hを入れています。



そしてCALL命令で、005C番地のルーチンをコールします。すると、E050番地からの、8バイトのスプライトパターンのデータが、スプライトジェネレータテーブルの3800番地に送られて、スプライトパターン番号0に定義されます。



繰り返えす回数をBレジスタに入れる

斜めにスプライトを移動させるときも、178頁で右にスプライトを移動させたときとおなじように、繰り返えしの命令であるＤＪＮＺ命令で、スプライトの表示を繰り返えします。ＤＪＮＺ命令で繰り返えすときは、Ｂレジスタに繰り返えす回数を入れると決まっていますから、繰り返えす回数をＢレジスタに入れます。

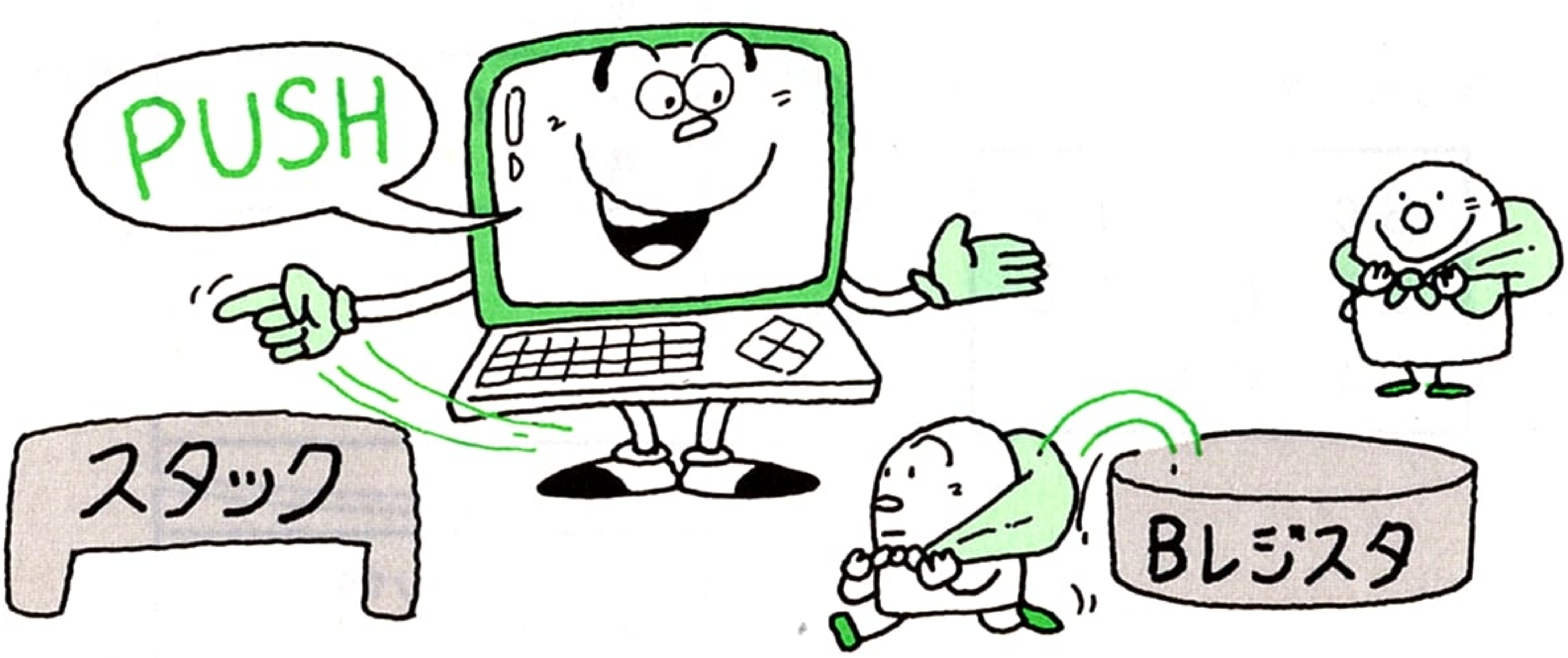
```
E00C  LD  B, A0H
```
←繰り返えす回数をBレジスタに入れる

ＡＯＨは10進数で160なので、画面の左上から右下までスプライトを移動させるには、ＤＪＮＺ命令で、スプライトの表示を160回繰り返えすことになります。

ＢレジスタにＡＯＨを入れたらＰＵＳＨ命令で、Ｂレジスタの内容ＡＯＨをスタック（244頁参照）に退避（たいひ）させます。なぜならこのあとで、Ｂレジスタにほかの値を入れるからです。レジスタはベーシックの変数とおなじで、あとから値を入れると、まえの値が消えて、あとから入れた値になってしまいます。それでは、ＤＪＮＺ命令で繰り返えす回数がわからなくなってしまうので、スタックに退避させておくのです。

```
E00E  PUSH  BC
```
←繰り返えす回数をスタックに退避させる

ＰＵＳＨ命令のあとには、かならずＢＣというように、レジスタペアにしておきます（179頁参照）。



スプライト属性テーブルの指定

つぎは、スプライトパターン番号０に定義したスプライトパターンを、スプライト面に表示させるために、スプライト属性（ぞくせい）テーブルの指定をします。ここでは、スプライト面番号０のスプライト面の左上、つまり、行座標００Ｈ（10進数で０）、列座標３０Ｈ（10進数で48）の位置に、スプライトパターンを白で表示しますから、スプライト属性テーブルを指定するデータは、つぎのようになります。

```
E058  00    30    00            0F
      行座標 列座標 スプライト面番号 色
```

このデータは、スプライトパターンのデータのあとにつづけます。

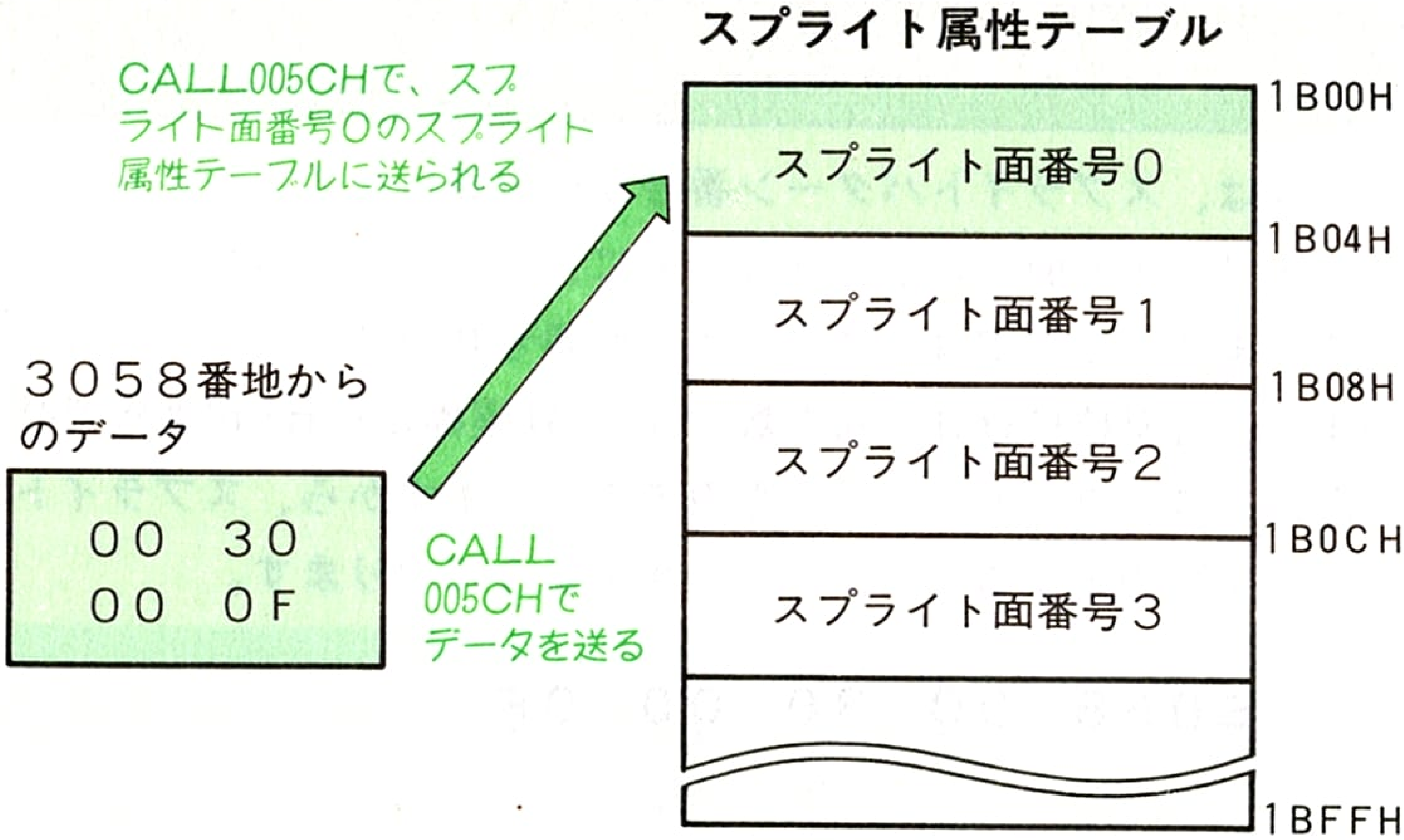
さて、スプライト属性テーブルの指定も、まえの場合とおなじように００５Ｃ番地のルーチンを使って、データをブロック転送して行いますから、ＨＬ、ＤＥ、ＢＣレジスタペアに、転送もとのメモリ番地、転送さきのメモリ番地、転送するデータのバイト数などの指定をしなければなりません。

```
E00F  LD   HL, E058H   ←データの最初のメモリ番地を入れる
E012  LD   DE, 1B00H   ←スプライト属性テーブルのメモリ番地を入れる
E015  LD   BC, 0004H   ←転送するデータのバイト数を入れる
E018  CALL 005CH       ←005C番地のルーチンをコールする
```

これらスプライト属性テーブルを指定する命令についても、180頁で説明しましたが、ここでも簡単に説明しておくことにします。

ＨＬレジスタには転送もと、つまり、スプライト属性テーブルを指定するデータが記憶されるメモリ番地を入れます。ここでは、さきに示したように、データはＥ０５８番地から割りあてられているので、ＨＬレジスタペアにはＥ０５８番地を入れています。

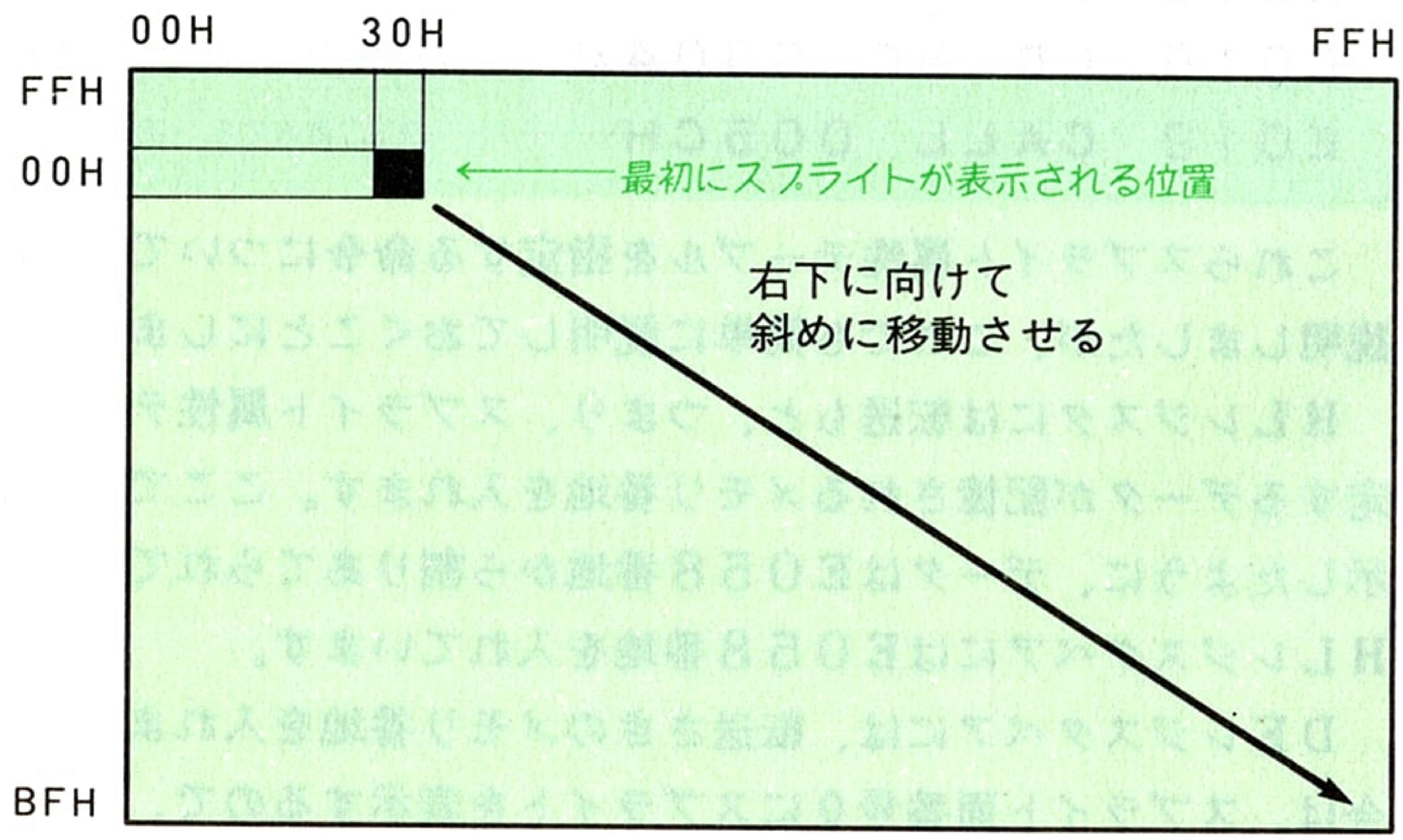
ＤＥレジスタペアには、転送さきのメモリ番地を入れます。この場合は、スプライト面番号０にスプライトを表示するので、スプライト



面番号0のスプライト属性テーブルにデータを送りますから、DEレジスタペアには1B00Hを入れます。

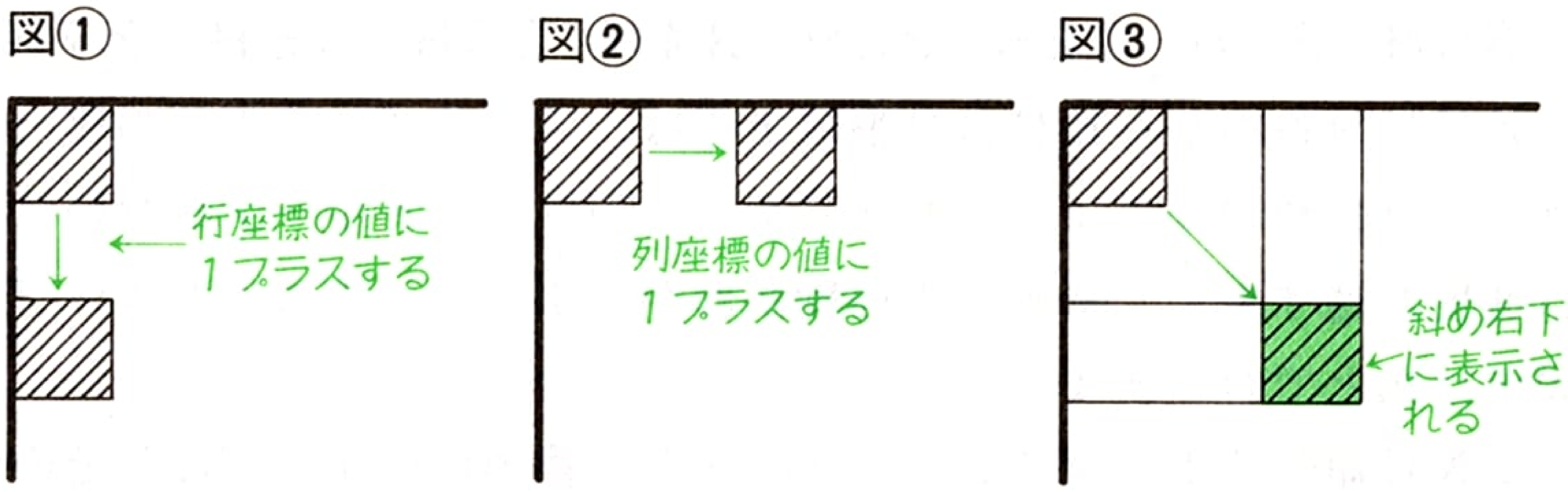
スプライト属性テーブルに送るデータは4バイトなので、BCレジスタペアには0004Hを入れています。

そして、CALL命令で005C番地のルーチンをコールします。これで、まえにスプライトパターン番号0に定義したスプライトパターンが、スプライト面番号0のスプライト面の、行座標00H、列座標30Hの位置に白で表示されます。



スプライトを斜めに移動させる

192頁の図に示した、最初にスプライトが表示される位置から、右下に向かって斜め(なな)めに移動させていくには、つぎの図①のように、行座標を指定する値に1をプラスして、スプライトを下に表示させます。そしてつぎに、図②のように列座標を指定する値に1をプラスして、スプライトを右に表示させます。すると図③に示すように、最初に表示された位置から斜め下に、スプライトが表示されることになります。



このことをＤＪＮＺ命令で繰り返えしていくと、スプライトは右下に向けて斜めに移動していきます。

行座標を指定する値に1をプラスするには、182頁で説明したようにＥ０５８番地の００Ｈを取りだしてＡレジスタに入れ、1プラスする命令であるＩＮＣ命令で、Ａレジスタの内容に1をプラスします。

```
E01B  LD  A, (E058H)  ←E058番地の値を取りだして、Aレジスタに入れる
E01E  INC A           ←Aレジスタの内容に1をプラスする
```

ＩＮＣ　Ａで、Ａレジスタの内容に1をプラスしたら、1をプラスしたＡレジスタの内容を、もとのＥ０５８番地に戻します。

```
E01F  LD  (E058H), A  ←もとのE058番地に戻す
```

つぎは、列座標を指定する３０ＨをＥ０５９番地から取りだしてＡレジスタに入れ、ＩＮＣ命令で1をプラスします。

```
E022  LD  A, (E059H)  ←E059番地の値を取りだしてAレジスタに入れる
E025  INC A           ←Aレジスタの内容に1プラスする
```

ＩＮＣ　Ａ命令でＡレジスタの内容に1をプラスしたら、1をプラスしたＡレジスタの内容を、もとのＥ０５９番地に戻します。

E026　LD　(E059H)，A←もとのメモリ番地に戻す

Ｅ０５８番地の行座標を指定する値と、Ｅ０５９番地の列座標を指定する値を取りだして、ＩＮＣ命令で１をプラスして、もとのメモリ番地に戻すということを繰り返えすと、Ｅ０５８番地の行座標を指定する値と、列座標を指定する値は、つぎのように増えていきます。

行座標を指定する値

00H　01H　02H　03H　04H　05H　06H　07H

列座標を指定する値

30H　31H　32H　33H　34H　35H　36H　37H

Ｅ０２９番地からＥ０３０番地までは、スプライトの表示の実行を遅らせるためのウエイトループです。このウエイトループがないと実行速度が速すぎて、斜めへの移動がわかりません。ウエイトループの説明はあとで行います。

ＩＮＣ命令で１プラスしてもとのメモリ番地に戻すたびに、その位置にスプライトを表示させるために、Ｅ０５８番地からの4バイトのデータをスプライト属性テーブルに繰り返えし送ります。

繰り返えす命令は、ＤＪＮＺ命令です。ＤＪＮＺ命令は、繰り返えしのために戻るとき、Ｂレジスタに入れた繰り返えしの回数から１を引いて戻ります。したがって、まえにＰＵＳＨ命令でスタックに退避させたＢレジスタの内容を、ＰＯＰ命令でスタックからＢレジスタに取りだしてきます。ＰＯＰ命令は、スタックからレジスタにデータを取りだしてくる命令です（184頁参照）。

ＰＯＰ命令のあとにも、レジスタペアをおきます。

E032　POP　BC←スタックから繰り返えしの回数を取り出す

これで、Ｂレジスタに入れた繰り返えしの回数から１を引くことができますから、つぎにＤＪＮＺ命令をおきます。

E033　DJNZ　E00EH←DJNZ命令でE00E番地に戻す

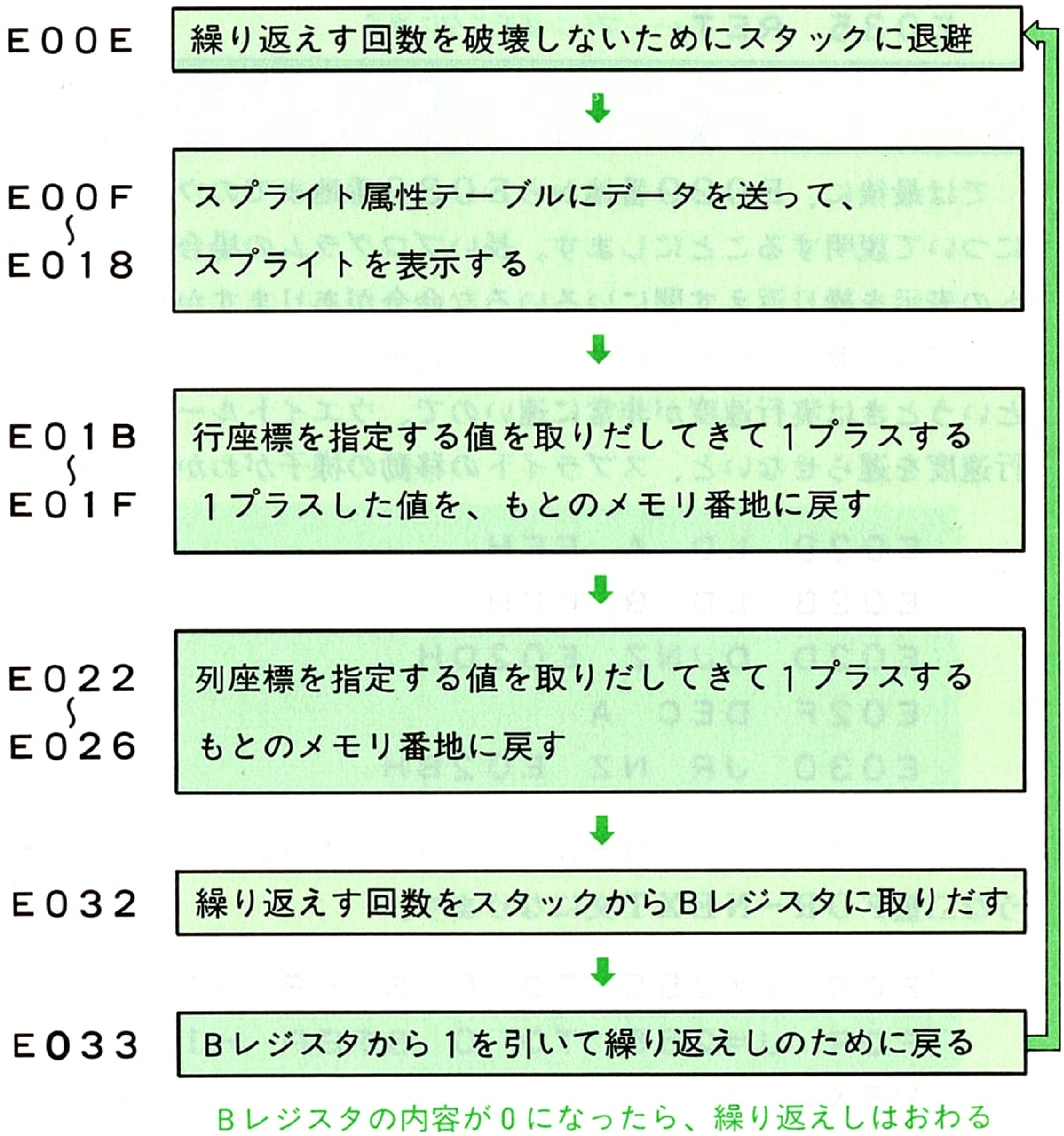
ＤＪＮＺ命令で繰り返えしのために戻るメモリ番地は、Ｅ００Ｅ番地です。Ｅ００Ｅ番地は、

E00E　PUSH　BC←DJNZ命令で戻る番地の命令

PUSH　BC命令ですから、ここから繰り返えすことになります。

なぜ、スプライト属性テーブルにデータを送る、最初の命令があるE00F番地に戻さないのかというと、E00F番地の命令のあとでBレジスタに新たにデータを入れて使っているからです。そのため、DJNZ命令で戻るたびに、PUSH命令でBレジスタの内容をスタックに退避させて、繰り返えしの回数が破壊されてわからなくならないようにするのです。

つぎに、DJNZ命令の繰り返えし方を図に示します。



ＤＪＮＺ命令で、図に示したように繰り返えすと、スプライトは画面の左上から右下に向かって斜めに移動していきます。

ＤＪＮＺ命令で繰り返えしのために戻るとき、Ｂレジスタの内容から１を引いていきます。したがって、ＡＯＨ(10進数で160) 回繰り返えすと、Ｂレジスタは０になります。Ｂレジスタの内容が０になると、ＤＪＮＺ命令による繰り返えしはおわって、つぎの命令の実行に移ります。このプログラムの場合は、つぎの命令はＲＥＴですから、マシン語モニタに戻ります。

```
E035  RET ←──マシン語モニタに戻る
```

ウエイトループ

では最後に、Ｅ０２９番地からＥ０３０番地までのウエイトループについて説明することにします。長いプログラムの場合は、スプライトの表示を繰り返えす間にいろいろな命令がありますから、ウエイトループは必要ないかもしれません。ただ単にスプライトを移動させるというときは実行速度が非常に速いので、ウエイトループを入れて実行速度を遅らせないと、スプライトの移動の様子がわかりません。

```
E029  LD  A, FFH
E02B  LD  B, FFH
E02D  DJNZ  E02DH
E02F  DEC  A
E030  JR  NZ  E02BH
```

このウエイトループをＭＳＸベーシックで書き表わすと、つぎのような二重ＦＯＲ～ＮＥＸＴ文になります。

```
FOR I=255 TO 0 STEP -1
FOR J=255 TO 0 STEP -1
NEXT J
NEXT I
```

二重FOR～NEXT文の働きについてはわかっていると思いますが、まず最初のFOR文を実行して、変数Iの値を255にします。変数Iの値を255にすると、つぎのFOR文を実行して、変数Jの値が255から0になるまでNEXT　Jとの間を繰り返えします。変数Jの値が0になると繰り返えしをやめて、NEXT　Iを実行して、最初のFOR文に戻ります。戻ったら変数Iの値を254にして、ふたたびつぎのFOR文を実行して、変数Jの値が255から0になるまで繰り返えします。このことを、最初のFOR文の変数Iが0になるまで繰り返えすというものです。これをマシン語で書くと、さきに示したようになるわけです。

A、Bレジスタに繰り返えす回数を入れる

では、マシン語のウエイトループの説明です。まず、

LD　A，FFH←――JR命令で繰り返えす回数を入れる

で、AレジスタにFFH（10進数で255)を入れます。これは、ウエイトループの最後にあるJR命令で繰り返えす回数です。JR命令については、あとで説明します。つぎに、

LD　B，FFH←――DJNZ命令で繰り返えす回数を入れる

で、BレジスタにFFHを入れます。これはウエイトループの3行目の、DJNZ命令で繰り返えす回数です。

DJNZ命令で繰り返えす

DJNZ　E02DH←――Bレジスタに入れたFFH回繰り返えす

で、Bレジスタに入れたFFHが0になるまで、DJNZ命令で繰り返えします。この場合、DJNZ命令のあとにあるメモリ番地と、DJNZ命令が記憶されるメモリ番地とがおなじですから、

E02D　DJNZ　E02DH

（E02D と E02DH：メモリ番地がおなじ）

つぎに示すように、DJNZ命令だけを繰り返えすことになります。

```
E02D  DJNZ  E02DH
```

ＤＪＮＺ命令で繰り返えすたびに、Ｂレジスタの内容から１を引いていきますから、ＤＪＮＺ命令でＦＦＨ回繰り返えすと、Ｂレジスタは０になります。するとＤＪＮＺ命令の繰り返えしはおわって、つぎの命令の実行に移ります。

DEC命令で１を引く

```
DEC  A ←――― Aレジスタの内容から１を引く
```

ＤＥＣ命令のＤＥＣは、英語のdecrement（デクリメント）の略記号です。英語のdecrementは、「減少」という意味ですが、マシン語のＤＥＣ命令は「１を引く」という働きです。

DEC命令＝１を引く命令

ここでは、ＤＥＣ命令のあとにＡレジスタがありますから、Ａレジスタの内容から１を引くことになります。Ａレジスタには、最初にＦＦＨを入れていますから、その値から１を引いていくことになります。

```
LD  A, FFH
  ↓          Aレジスタの内容から１を引く
DEC  A ←
```

JR命令で、LD　B，FFHを繰り返えす

ＤＥＣ　Ａ命令で、Ａレジスタの内容から１を引いたら、

```
JR  NZ  E02BH ←――― ゼロフラグが０のときE02B番地にジャンプ
```

を実行します。ＪＲ命令のＪＲは、英語のjump　relative（ジャンプ・リラティブ）の略記号です。英語の jump は「飛ぶ」、relative は「相対的な」という意味です。「相対的なジャンプ」命令などといっても、なんのことかわからなくて当然です。

ＪＲ命令は、ＭＳＸベーシックのＧＯＴＯ文とおなじで、指定した

メモリ番地にジャンプさせる命令です。

JR命令＝指定したメモリ番地にジャンプさせる命令

ただし、ＪＲ命令の場合は、アセンブリ言語をマシン語になおすとき、直接メモリ番地を指定するのではなくて、相対アドレス指定というメモリ番地の指定の仕方でメモリ番地を指定します。

アセンブリ言語　　ＪＲ　ＮＺ　Ｅ０２ＢＨ

↙

マシン語　　　　　２０　Ｆ９←相対アドレス指定によるメモリ番地

このことから、ＪＲ命令を「相対的なジャンプ」、つまり「相対的ジャンプ」命令というのです。相対アドレス指定の仕方については230頁で説明します。

ところで、ＪＲ命令にはもうひとつ、ＮＺというものがおいてあります。このＮＺは条件です。

ＪＲ　ＮＺ　Ｅ０２ＢＨ

↑ゼロフラグが０のとき、という条件

ＮＺはどういう条件かというと、ゼロフラグが０のときという条件です。ゼロフラグについては236頁で説明しますが、ゼロフラグは、演算の結果が０でないとき０、演算の結果が０のとき１になります。

ゼロフラグが０＝演算の結果が０でないとき
ゼロフラグが１＝演算の結果が０のとき

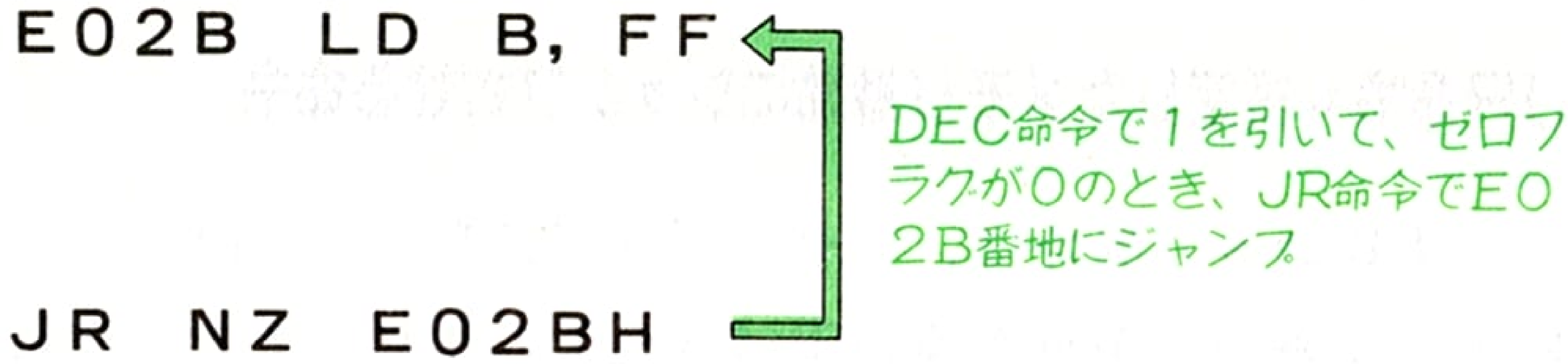
ゼロフラグが０になったり、１になったりする演算は、この場合、

ＤＥＣ　Ａ

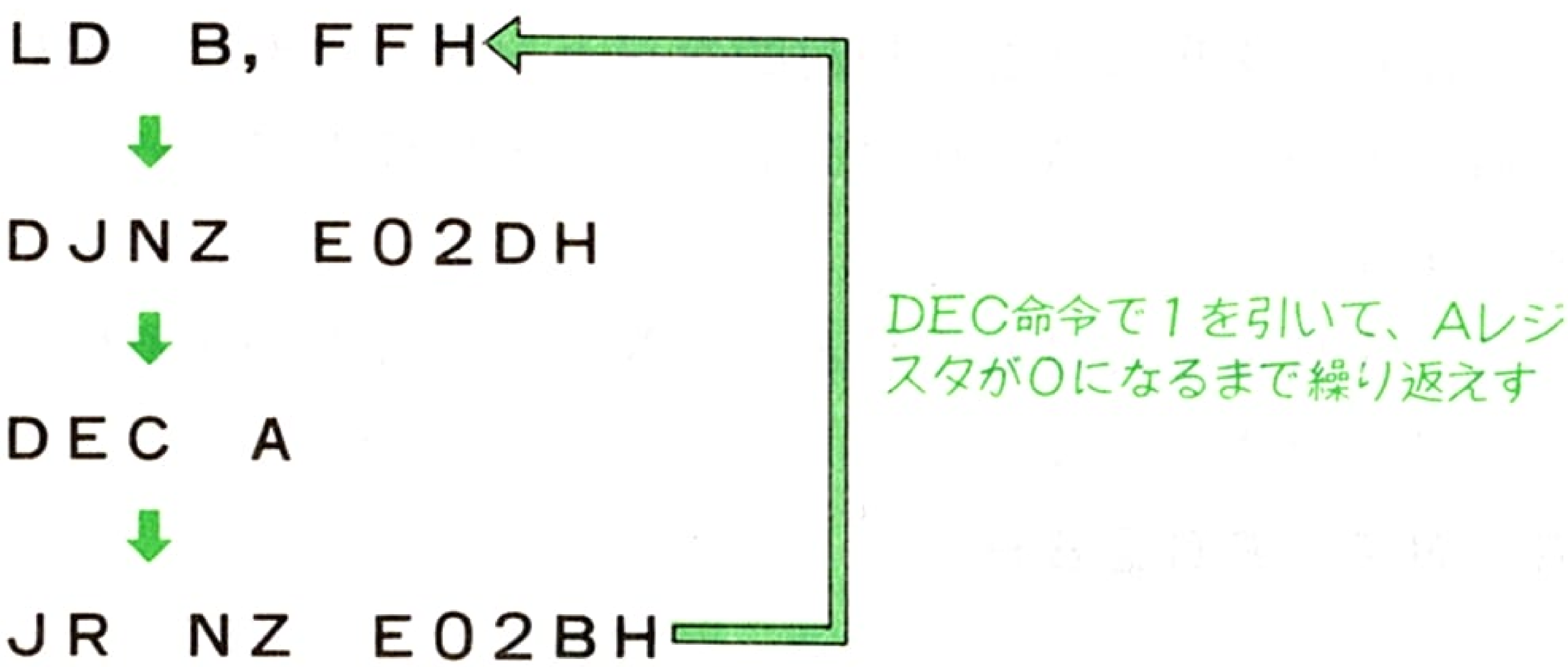
です。つまり、Ａレジスタから１を引いた結果、Ａレジスタの内容が０にならないとき、ゼロフラグは０、Ａレジスタから１を引いた結果、Ａレジスタの内容が０になると、ゼロフラグは１になります。

ＡレジスタにはＦＦＨ（10進数で255）を入れていますから、ＤＥＣ命令で255回繰り返えさないと、Ａレジスタは０になりません。したがって、Ａレジスタが０にならない間は、ＮＺの条件が成立して、Ｊ

R命令に指定されているE02B番地にジャンプします。



E02B番地の命令はLD B, FFHですから、ふたたびBレジスタにFFHを入れて、DJNZ命令で繰り返えします。繰り返えしがおわると、DEC命令で1を引き、JR命令でまたE02B番地にジャンプします。



このことを繰り返えすと、Aレジスタの内容は0になりますから、ゼロフラグは1になります。ゼロフラグが1になると、JR命令のNZの条件は成立しませんから、E02B番地にジャンプすることをやめて、つぎの命令の実行に移ります。これがウエイトループの働きです。



複数個のスプライトを表示する

3個のスプライトの色を変えて、横に表示

164頁で、スプライトを1個、スプライト面に表示しました。ここでは複数個のスプライトを、スプライト面に表示することにします。

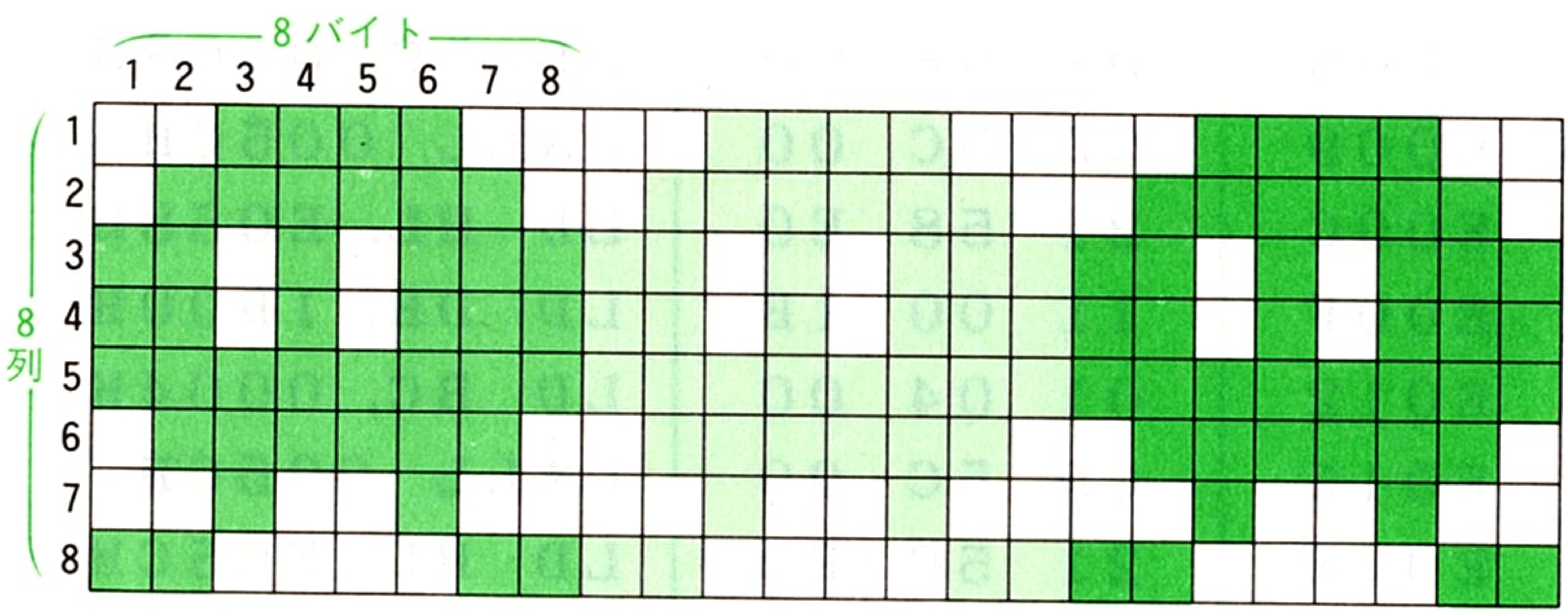
まず、プログラムを示します。

メモリ番地	マシン語	アセンブリ言語
E000	21 50 E0	LD HL, E050H
E003	11 00 38	LD DE, 3800H
E006	01 08 00	LD BC, 0008H
E009	CD 5C 00	CALL 005CH
E00C	21 58 E0	LD HL, E058H
E00F	11 00 1B	LD DE, 1B00H
E012	01 04 00	LD BC, 0004H
E015	CD 5C 00	CALL 005CH
E018	21 5C E0	LD HL, E05CH
E01B	11 04 1B	LD DE 1B04H
E01E	01 04 00	LD BC, 0004H
E021	CD 5C 00	CALL 005CH
E024	21 60 E0	LD HL, E060H
E027	11 08 1B	LD DE, 1B08H
E02A	01 04 00	LD BC, 0004H
E02D	CD 5C 00	CALL 005CH
E030	C9	RET

E050	3C　7E　D7	〝データ〟3C　7E　D7
E053	D7　FF　7E	D7　FF　7E　24　C3
E056	24　C3　60	〝データ〟60　80　00
E059	80　00　0F	0F
E05C	60　88　00	〝データ〟60　88　00
E05F	09　60　90	09
E062	00　0B	〝データ〟60　90　00
		0B

スプライトパターンの定義

ここではつぎの図に示すように、おなじ形のスプライトを、色を変えて横に3個表示することにします。



おなじ形のスプライトを3個表示するときは、スプライトパターンのデータは3個分必要ありません。1個分のデータでよいのです。したがって、この場合のスプライトパターンのデータは、つぎのように1個分のデータ（8バイト）となります。

E050　3C　7E　D7　D7　FF　7E　24　C3

スプライトをスプライト面に表示するために、この8バイトのスプライトパターンのデータを、スプライトジェネレータテーブルに送って、スプライトパターンの定義をします。

```
E000 LD HL, E050H
E003 LD DE, 3800H
E006 LD BC, 0008H
E009 CALL 005CH
```

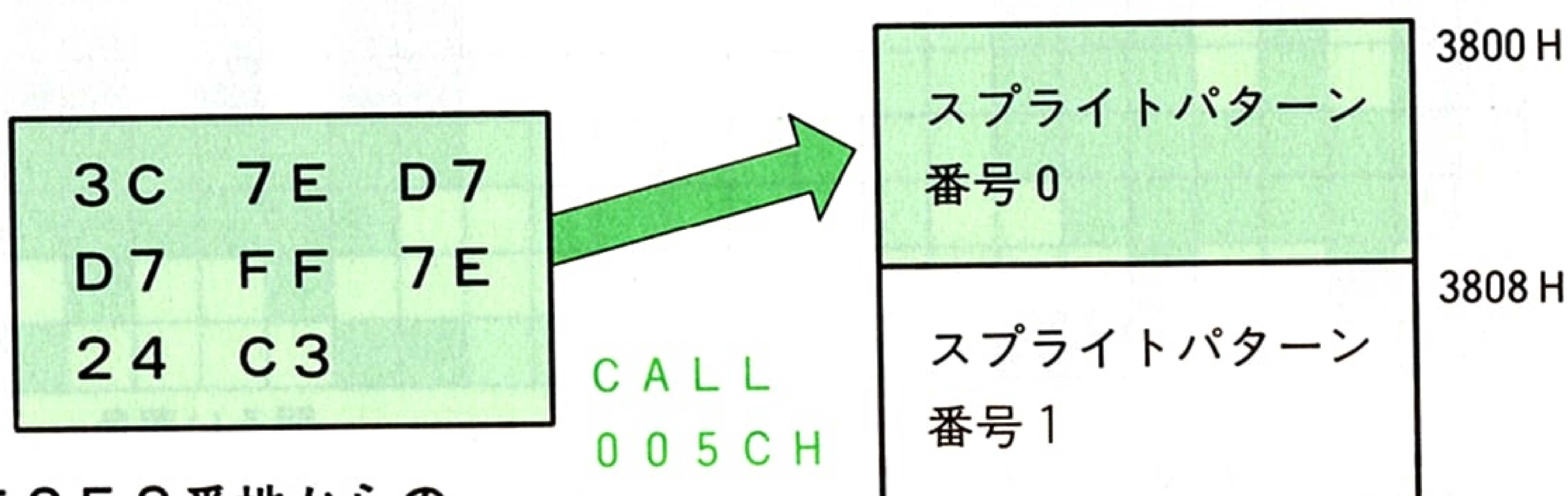
スプライトパターンをスプライトジェネレータテーブルに定義するには、005C番地にあるVDPのVRAMにブロック転送するルーチンを使って行いますから、HLレジスタペアに、データが記憶されているメモリ番地を入れ、DEレジスタペアに、転送するさきのスプライトジェネレータテーブルのメモリ番地を入れ、BCレジスタペアには、転送するデータのバイト数を入れます。

この場合は、スプライトパターンのデータはE050番地からなので、HLレジスタペアにはE050番地を入れています。

スプライトパターンは、スプライトパターン番号0に定義するので、DEレジスタペアには、スプライトジェネレータテーブルの3800番地を入れています。

スプライトジェネレータテーブルに送る、スプライトパターンのデータの長さ（バイト数）は8バイトなので、BCレジスタペアには0008Hを入れています。

このようにしてから、CALL命令で005C番地のルーチンをコールすると、つぎの図に示すようにE050番地からの、8バイトのスプライトパターンのデータがスプライトジェネレータテーブルの3800番地に送られて、スプライトパターン番号0に定義されます。



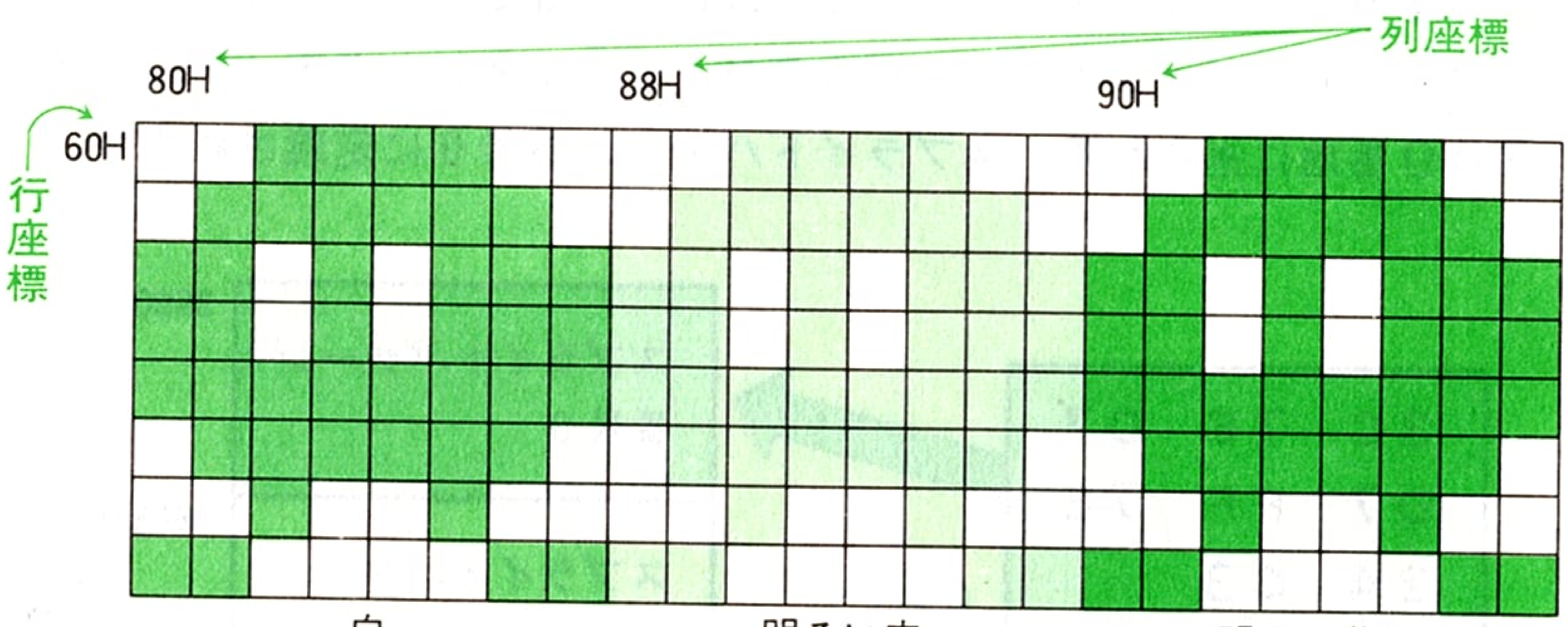
E050番地からの
スプライトパターンのデータ　　スプライトジェネレータテーブル

スプライト属性テーブルの指定

スプライトジェネレータテーブルにスプライトパターンを定義したら、つぎはスプライトを表示するために、スプライト属性テーブルにスプライトを表示する行座標（垂直方向)、列座標（水平方向)、スプライトジェネレータテーブルに定義したスプライトパターン番号、表示するスプライトの色のデータを送って指定します。

スプライトパターンを定義するときは、スプライトパターンのデータは1個分のデータですみましたが、スプライト属性テーブルを指定するときは、表示するスプライトの数だけデータが必要になります。なぜなら、スプライト面には1個のスプライトしか表示することができませんし、また、スプライトを表示する位置がそれぞれ異なるからです。加えて表示する色もそれぞれ異なるからです。

ここではつぎの図に示すように、スプライト面番号0から2までのスプライト面に、横に3個のスプライトを並べて表示することにします。1個目のスプライトは行座標６０H、列座標８０Hの位置に、白で表示します。2個目のスプライトは行座標６０H、列座標８８Hの位置に、明るい赤で表示します。3個目のスプライトは行座標６０H、列座標９０Hの位置に、明るい黄色で表示することにします。



したがって、1個目のスプライトの、スプライト属性テーブルを指定するデータは、つぎのようになります。

E058　60　80　00　0F

行座標　列座標　スプライトパターン番号　色

2個目のスプライトの、スプライト属性テーブルを指定するデータは、つぎのようになります。

E05C　60　88　00　09

3個目のスプライトの、スプライト属性テーブルを指定するデータは、つぎのようになります。

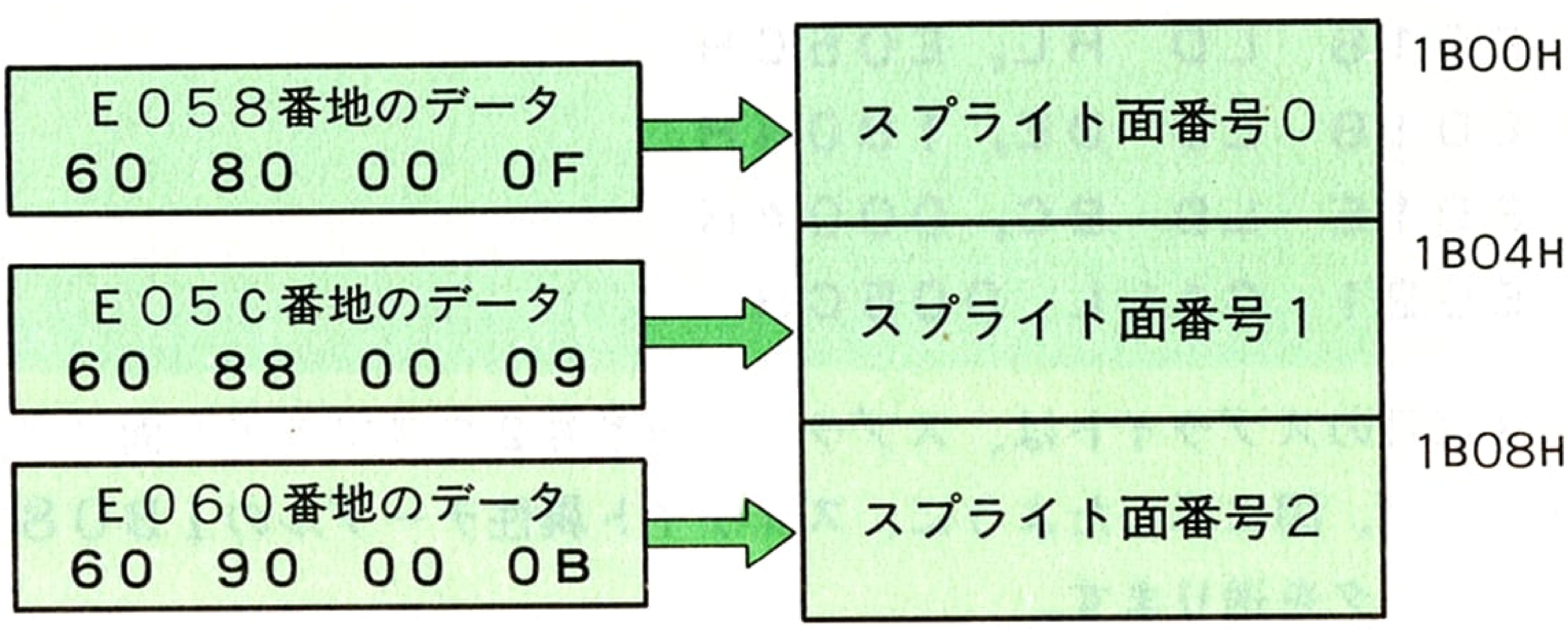
E060　60　90　00　0B

スプライト属性テーブルを指定するには、このデータをVDPのVRAMにブロック転送する005C番地のルーチンを使います。005C番地のルーチンを使うには、さきにスプライトジェネレータテーブルにデータを送ったときとおなじように、HLレジスタペアにデータが記憶されているメモリ番地を入れ、DEレジスタペアにデータを転送するさきのメモリ番地を入れ、BCレジスタペアに転送するデータのバイト数を入れなければなりません。

データが記憶されているメモリ番地は、さきに示したように1個目のデータがE058番地、2個目のデータがE05C番地、3個目がE060番地です。転送するデータのバイト数は、さきに示したようにそれぞれ4バイトです。

転送するさきのスプライト属性テーブルのメモリ番地を、つぎに図に示します。

スプライト属性テーブル



１個目のスプライトは、スプライト面番号０のスプライト面に表示するので、図に示したように、スプライト属性テーブルの１Ｂ００番地にデータを送りますから、転送さきのメモリ番地は１Ｂ００番地になります。

転送するデータが記憶されているメモリ番地はＥ０５８番地、転送するデータのバイト数は４バイトですから、１個目のスプライトのスプライト属性テーブルを指定する命令は、つぎのようになります。

```
E00C LD HL, E058H ←データが記憶されているメモリ番地を入れる
E00F LD DE, 1B00H ←転送するさきのスプライト属性テーブルのメモリ番地を入れる
E012 LD BC, 0004H ←転送するデータのバイト数を入れる
E015 CALL 005CH  ←005C番地のルーチンをコールする
```

２個目のスプライトは、スプライト面番号１のスプライト面に表示するので、図に示したように、スプライト属性テーブルの１Ｂ０４番地にデータを送りますから、転送さきのメモリ番地は１Ｂ０４番地になります。

転送するデータが記憶されているメモリ番地はＥ０５Ｃ番地、転送するデータのバイト数は４バイトですから、２個目のスプライトのスプライト属性テーブルを指定する命令は、つぎのようになります。

```
E018 LD HL, E05CH
E01B LD DE, 1B04H
E01E LD BC, 0004H
E021 CALL 005CH
```

３個目のスプライトは、スプライト面番号２のスプライト面に表示するので、図に示したように、スプライト属性テーブルの１Ｂ０８番地にデータを送ります。

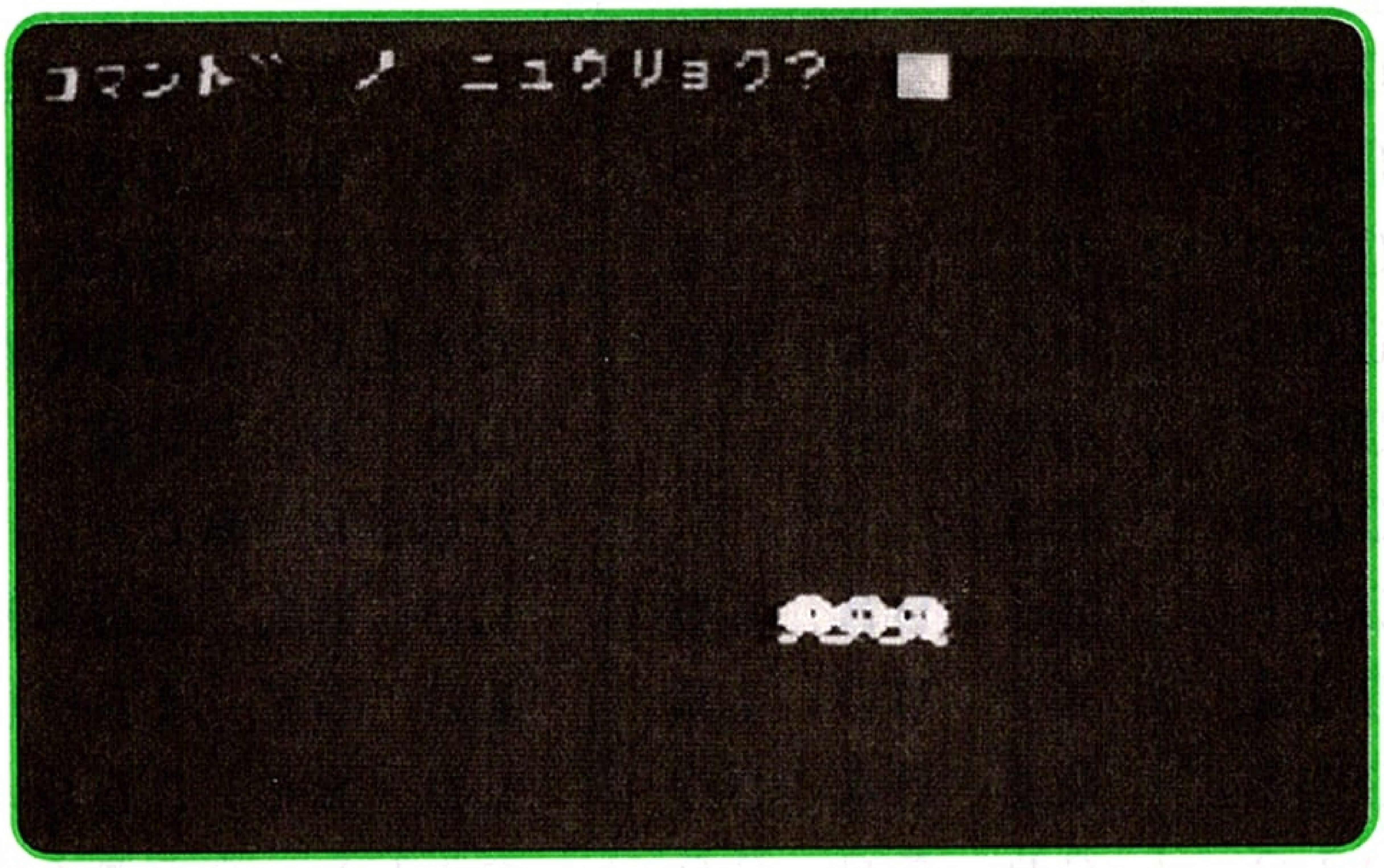
転送するデータが記憶されているメモリ番地はＥ０６０番地、転送するデータのバイト数は4バイトですから、3個目のスプライトのスプライト属性テーブルを指定する命令は、つぎのようになります。

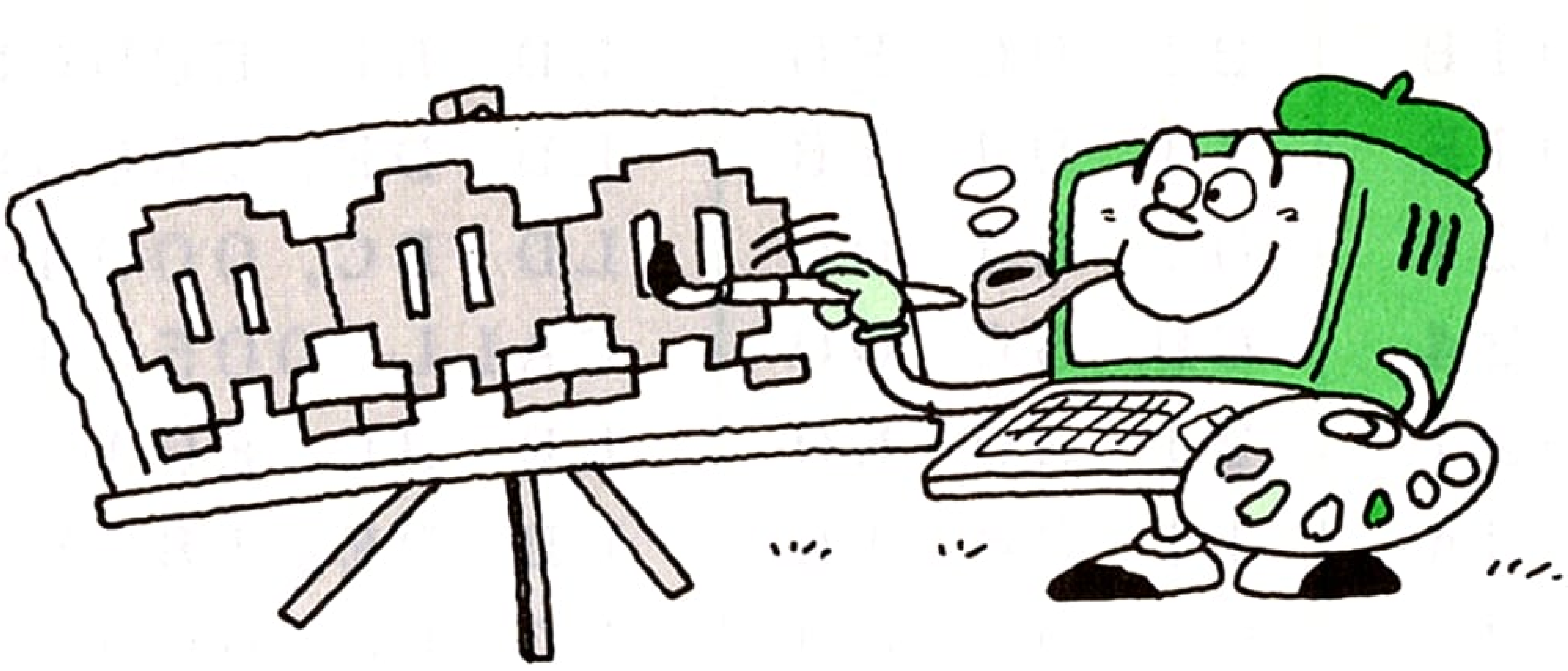
```
E024  LD  HL, E060H
E027  LD  DE, 1B08H
E02A  LD  BC, 0004H
E02D  CALL  005CH
```

最後にマシン語モニタに戻るＲＥＴをおくと、3個のスプライトをスプライト面に表示することができます。

RET ←――――マシン語モニタに戻る

つぎに、このマシン語プログラムの実行結果を示します。





複数個のスプライトの移動

左上から右下に向けて、斜めに移動させる

複数個のスプライトの移動には、右への移動、左への移動、また下への移動、上への移動などいろいろありますが、ここでは画面の左上から右下に向けての、斜めの移動を取りあげることにします。

最初にプログラムを示します。

メモリ番地	マ シ ン 語	アセンブリ言語
E000	21 90 E0	LD HL, E090H
E003	11 00 38	LD DE, 3800H
E006	01 08 00	LD BC, 0008H
E009	CD 5C 00	CALL 005CH
E00C	06 70	LD B, 70H
E00E	C5	PUSH BC
E00F	21 98 E0	LD HL, E098H
E012	11 00 1B	LD DE, 1B00H
E015	01 04 00	LD BC, 0004H
E018	CD 5C 00	CALL 005CH
E01B	21 9C E0	LD HL, E09CH
E01E	11 04 1B	LD DE, 1B04H
E021	01 04 00	LD BC, 0004H
E024	CD 5C 00	CALL 005CH
E027	21 A0 E0	LD HL, E0A0H
E02A	11 08 1B	LD DE, 1B08H
E02D	01 04 00	LD BC, 0004H

E030	CD 5C 00	CALL 005CH
E033	3A 98 E0	LD A, (E098H)
E036	3C	INC A
E037	32 98 E0	LD (E098H), A
E03A	3A 99 E0	LD A, (E099H)
E03D	3C	INC A
E03E	3C	INC A
E03F	32 99 E0	LD (E099H), A
E042	3E FF	LD A, FFH
E044	06 0F	LD B, 0FH
E046	10 FE	DJNZ E046H
E048	3D	DEC A
E049	20 F9	JR NZ E044H
E04B	3A 98 E0	LD A, (E098H)
E04E	32 9C E0	LD (E09CH), A
E051	32 A0 E0	LD (E0A0H), A
E054	3A 99 E0	LD A, (E099H)
E057	16 08	LD D, 08H
E059	82	ADD A, D
E05A	32 9D E0	LD (E09DH), A
E05D	82	ADD A, D
E05E	32 A1 E0	LD (E0A1H), A
E061	C1	POP BC
E062	10 AA	DJNZ E00EH
E064	C9	RET
E090	3C 7E D7	"データ" 3C 7E
E093	D7 FF 7E	D7 D7 FF 7E

E096 E099 E09C E09F E0A2	24 C3 00 00 00 0F 00 08 00 09 00 10 00 0B	24 C3 〝データ〞00 00 00 0F 〝データ〞00 08 00 09 〝データ〞00 10 00 0B

スプライトパターンの定義

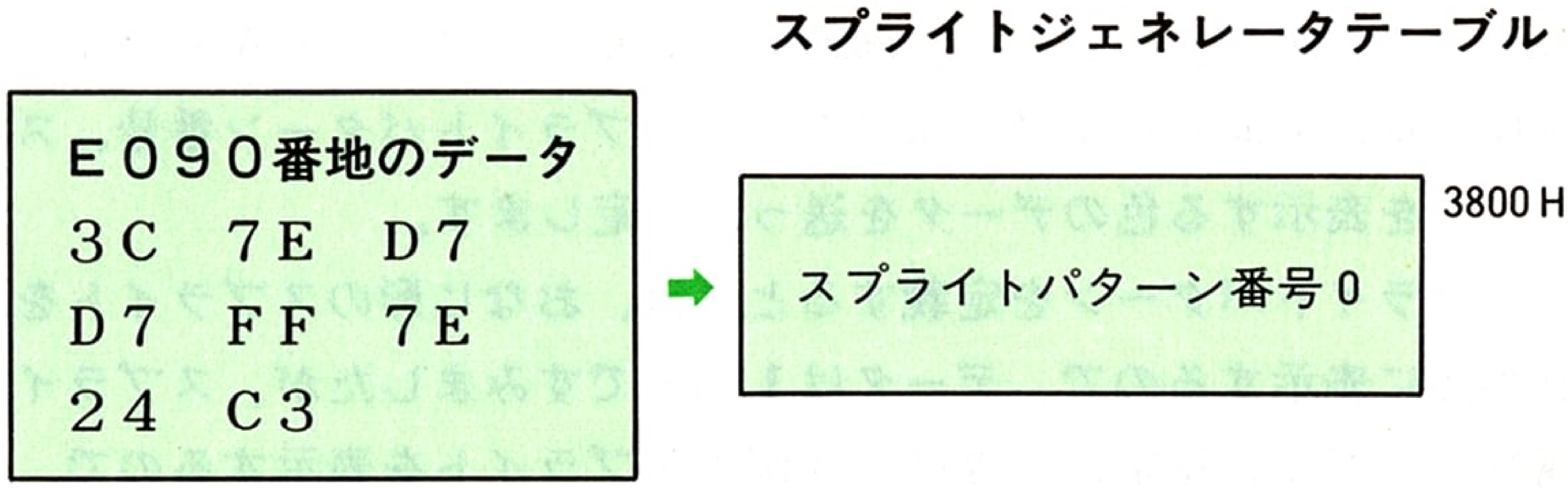
ここでは、おなじ形のスプライトを3個画面の左上に表示して、画面の右下に斜めに移動させることにします。例によって、まず表示するスプライトのパターンをスプライトジェネレータテーブルに定義します。おなじ形のスプライトを3個表示するときは、202頁で説明したように、1個分のスプライトパターンのデータを、スプライトジェネレータテーブルに送って定義すればよいので、スプライトパターンのデータは、つぎのように1個分のデータ（8バイト）になります。

E090 3C 7E D7 D7 FF 7E 24 C3

このスプライトパターンのデータをスプライトジェネレータテーブルに送って定義するには、005C番地のルーチンを使って行いますから、HLレジスタペアにデータが記憶されているメモリ番地、DEレジスタペアにデータを転送するさきのメモリ番地、BCレジスタペアに転送するデータのバイト数を入れて、005C番地をコールします。

```
E000 LD   HL,E090H ←データが記憶されている最初のメモリ番地を入れる
E003 LD   DE,3800H ←データを転送するスプライトジェネレータテーブルのメモリ番地を入れる
E006 LD   BC,0008H ←転送するデータのバイト数を入れる
E009 CALL 005CH    ←005C番地のルーチンをコールする
```

これでつぎの図に示すように、E090番地から記憶されているスプライトパターンのデータが、スプライトジェネレータテーブルの3800番地に送られて、スプライトパターン番号0に定義されます。



繰り返えす回数を、Bレジスタに入れる

187頁で、1個のスプライトを画面の左上から右下に向けて斜めに移動させましたが、その場合、DJNZ命令でスプライトを繰り返えし表示させて、移動させました。そして、DJNZ命令で繰り返えす回数をBレジスタに入れました。この場合も、DJNZ命令で3個のスプライトを繰り返えし表示させて移動させますから、Bレジスタに繰り返えす回数を入れます。

E00C　LD　B, 70H ←DJNZ命令で繰り返えす回数を入れる

70Hは、10進数で112ですから、DJNZ命令で112回繰り返えすことになります。

DJNZ命令で繰り返えす回数をBレジスタに入れたら、PUSH命令で、Bレジスタの内容をスタックに退避させます。Bレジスタの内容をスタックに退避させるのは、このあとで、Bレジスタにほかの値を入れて使うからです。

E00E　PUSH　BC ←Bレジスタの内容をスタックに退避

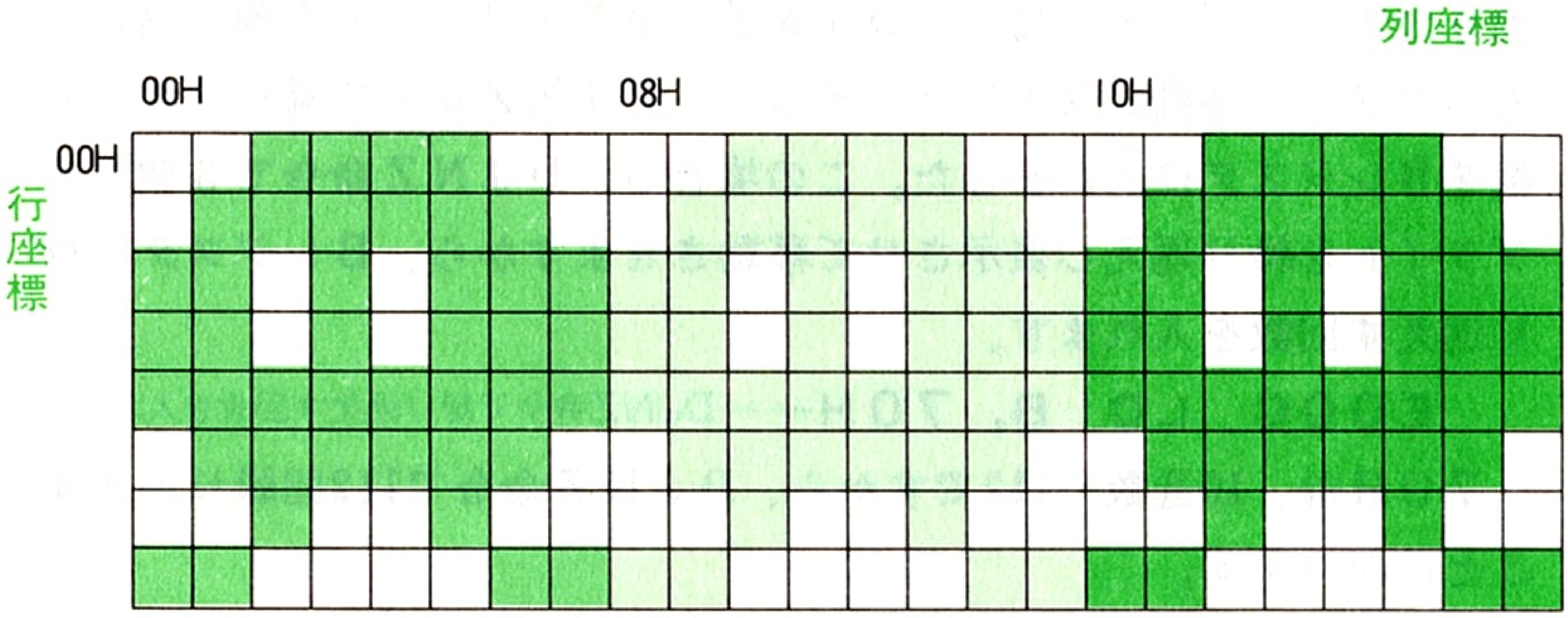
スタックについては244頁、PUSH命令については179頁を参照してください。

スプライト属性テーブルの指定

つぎは、スプライトパターン番号0に定義したスプライトを画面に表示するために、スプライト属性テーブルにスプライトを表示する行座標（垂直方向）、列座標（水平方向）、スプライトパターン番号、スプライトを表示する色のデータを送って指定します。

スプライトパターンを定義するときは、おなじ形のスプライトを3個画面に表示するので、データは1個分ですみましたが、スプライト属性テーブルの指定のときは、3個のスプライトを表示するので、データは3個分必要になります。

ここでは3個のスプライトを、つぎの図に示す位置に表示して、それから、画面の右下に向けて斜めに移動させていきます。



1個目のスプライトを最初に表示する位置は、行座標００H、列座標００Hとなります。表示する色は白にします。したがって、スプライト属性テーブルを指定するデータは、つぎのようになります。

E098　00　00　00　0F ←色

行座標　列座標　スプライトパターン番号

2個目のスプライトを最初に表示する位置は、行座標００H、列座標０８Hとなります。表示する色は明るい赤にします。したがって、スプライト属性テーブルを指定するデータは、つぎのようになります。

E09C　00　08　00　09

3個目のスプライトを最初に表示する位置は、行座標００H、列座標

標１０Hとなります。表示する色はあかるい黄色にします。したがって、スプライト属性テーブルを指定するデータは、つぎのようになります。

```
E0A0  00  10  00  0B
```

さて、スプライト属性テーブルの指定も、００５C番地のデータをブロック転送するルーチンを使って行いますから、HLレジスタペアには、データが記憶されるメモリ番地を入れます。DEレジスタペアには、データを転送するさきのスプライト属性テーブルのメモリ番地を入れます。BCレジスタペアには、転送するデータのバイト数を入れます。そして、このデータをVDPのVRAMにブロック転送するために００５C番地のルーチンをコールします。

まず、1個目のスプライトのデータをスプライト属性テーブルに送ります。1個目のスプライトのデータはE０９８番地から記憶されていますから、HLレジスタペアにはE０９８Hを入れます。

1個目のスプライトは、スプライト面番号0のスプライト面に表示するので、データを転送するさきのスプライト属性テーブルのメモリ番地は１B００番地です。したがって、DEレジスタペアには１B００Hを入れます。

スプライト属性テーブルに送るデータのバイト数は4バイトですから、BCレジスタペアには０００４Hを入れます。

```
E00F  LD  HL, E098H
E012  LD  DE, 1B00H
E015  LD  BC, 0004H
E018  CALL  005CH
```

つぎは、2個目のスプライトのデータをスプライト属性テーブルに送ります。

2個目のスプライトのデータはE０９C番地から記憶されていますから、HLレジスタペアにはE０９CHを入れます。

2個目のスプライトは、スプライト面番号1のスプライト面に表示

するので、データを転送するさきのスプライト属性テーブルのメモリ番地は１Ｂ０４番地です。したがって、ＤＥレジスタペアには１Ｂ０４Ｈを入れます。

１Ｂ０４番地のスプライト属性テーブルに送るデータのバイト数は4バイトですから、ＢＣレジスタペアに０００４Ｈを入れます。

```
E01B  LD  HL, E09CH
E01E  LD  DE, 1B04H
E021  LD  BC, 0004H
E024  CALL  005CH
```

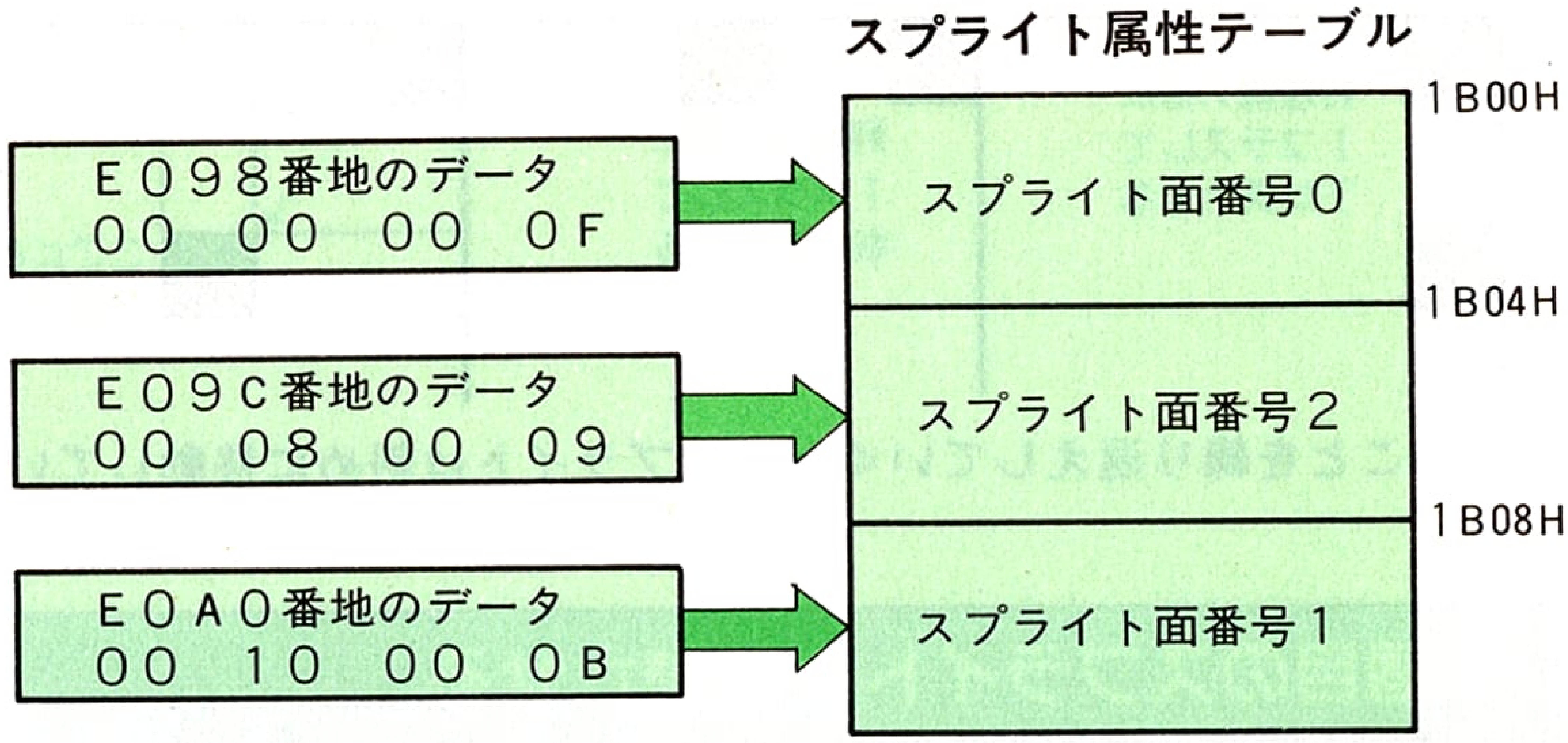
最後は3個目のスプライトのデータをスプライト属性テーブルに送ります。3個目のスプライトのデータはＥ０Ａ０番地から記憶されていますから、ＨＬレジスタペアにはＥ０Ａ０Ｈを入れます。

3個目のスプライトは、スプライト面番号2のスプライト面に表示するので、データを転送するさきのスプライト属性テーブルのメモリ番地は１Ｂ０８番地です。したがって、ＤＥレジスタペアには１Ｂ０８番地を入れます。

１Ｂ０８番地のスプライト属性テーブルに送るデータのバイト数は4バイトですから、ＢＣレジスタペアには０００４Ｈを入れます。

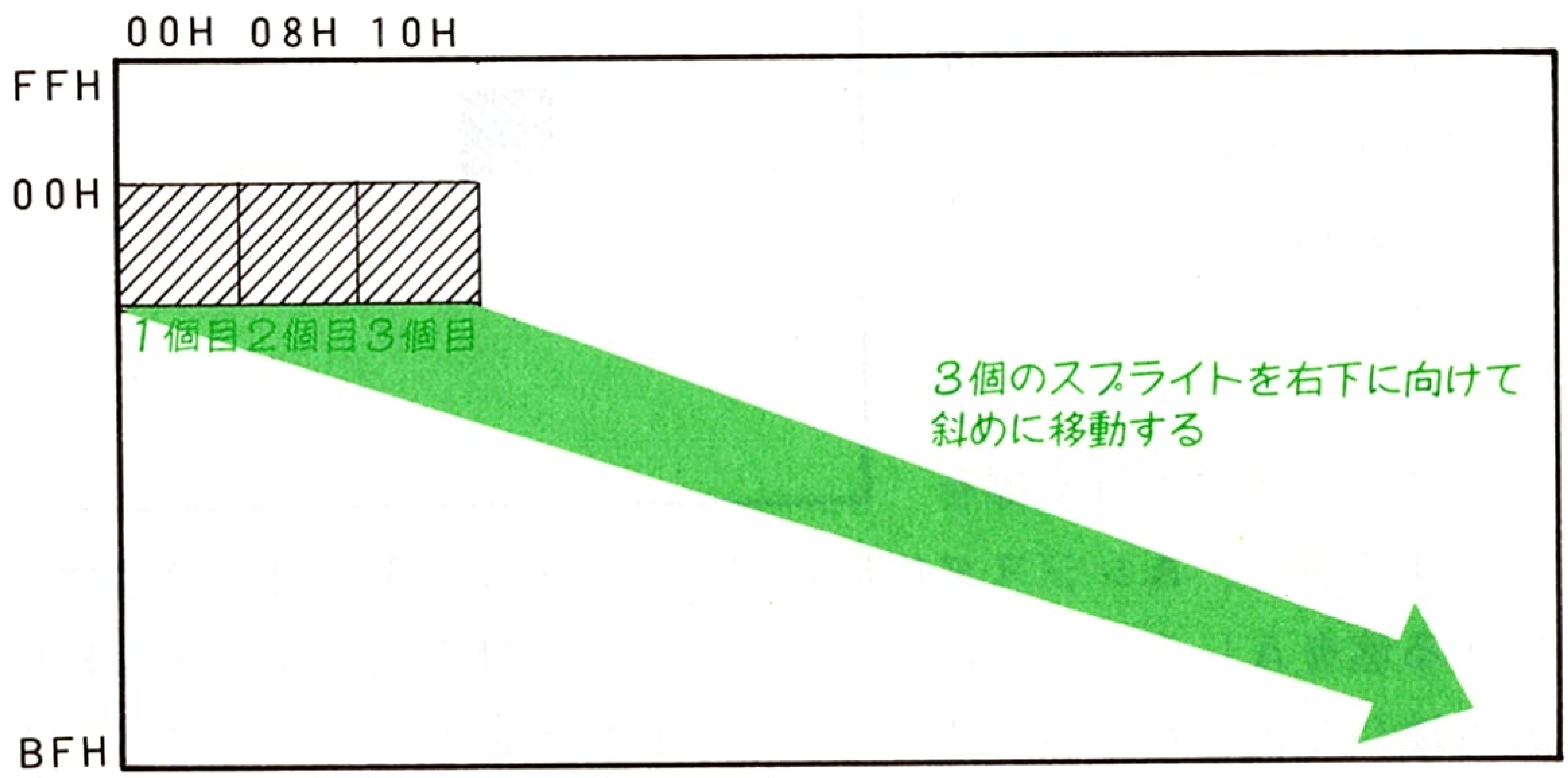
```
E027  LD  HL, E0A0H
E02A  LD  DE, 1B08H
E02D  LD  BC, 0004H
E030  CALL  005CH
```

これで、3個のスプライトを画面に表示するためのデータが、ＶＤＰのＶＲＡＭにブロック転送する００５Ｃ番地のルーチンで、つぎの図に示すように、１Ｂ００番地、１Ｂ０４番地、１Ｂ０８番地のスプライト属性テーブルに送られて、スプライトパターン番号0に定義したスプライトパターンが、スプライト面番号0から2までのスプライト面に表示されることになります。



3個のスプライトを移動させる

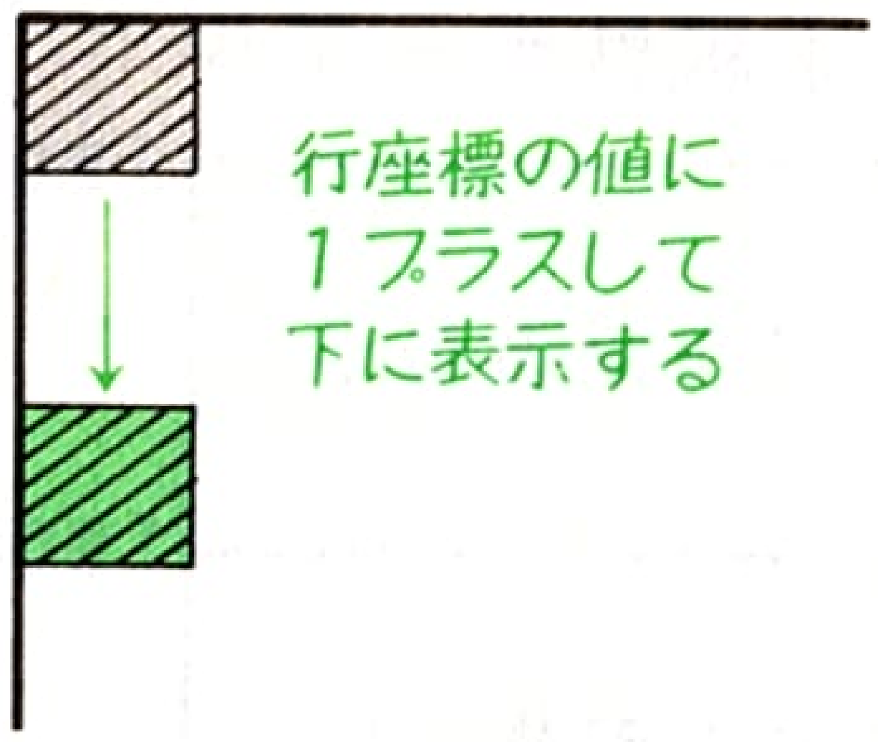
これで3個のスプライトは、スプライト面番号0から2までのスプライト面に、つぎの図に示す位置に表示されますから、この3個のスプライトを右下に向けて移動させていきます。



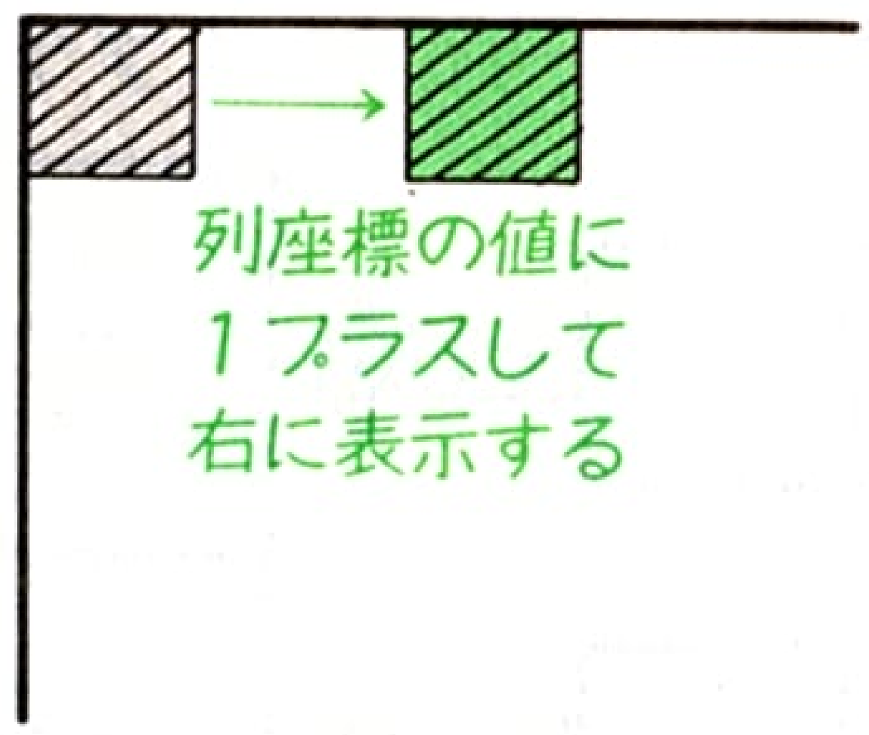
1個のスプライトを右下に向けて移動させていくには、つぎの頁の図で示すように、まず行座標の値に1をプラスして、スプライトを下に表示させます(216頁図①)。つぎに列座標の値に1をプラスして、スプライトを右に表示させます(図②)。

すると、スプライトは、それまで表示されていた位置より斜め下の位置に表示されることになります(図③)。

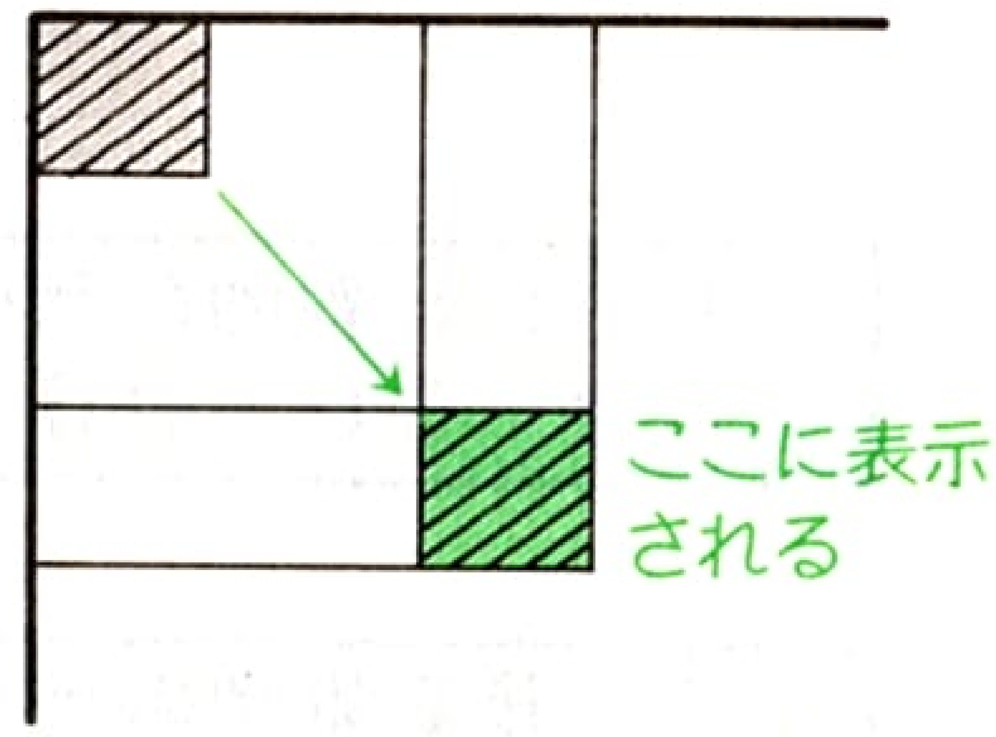
図①



図②



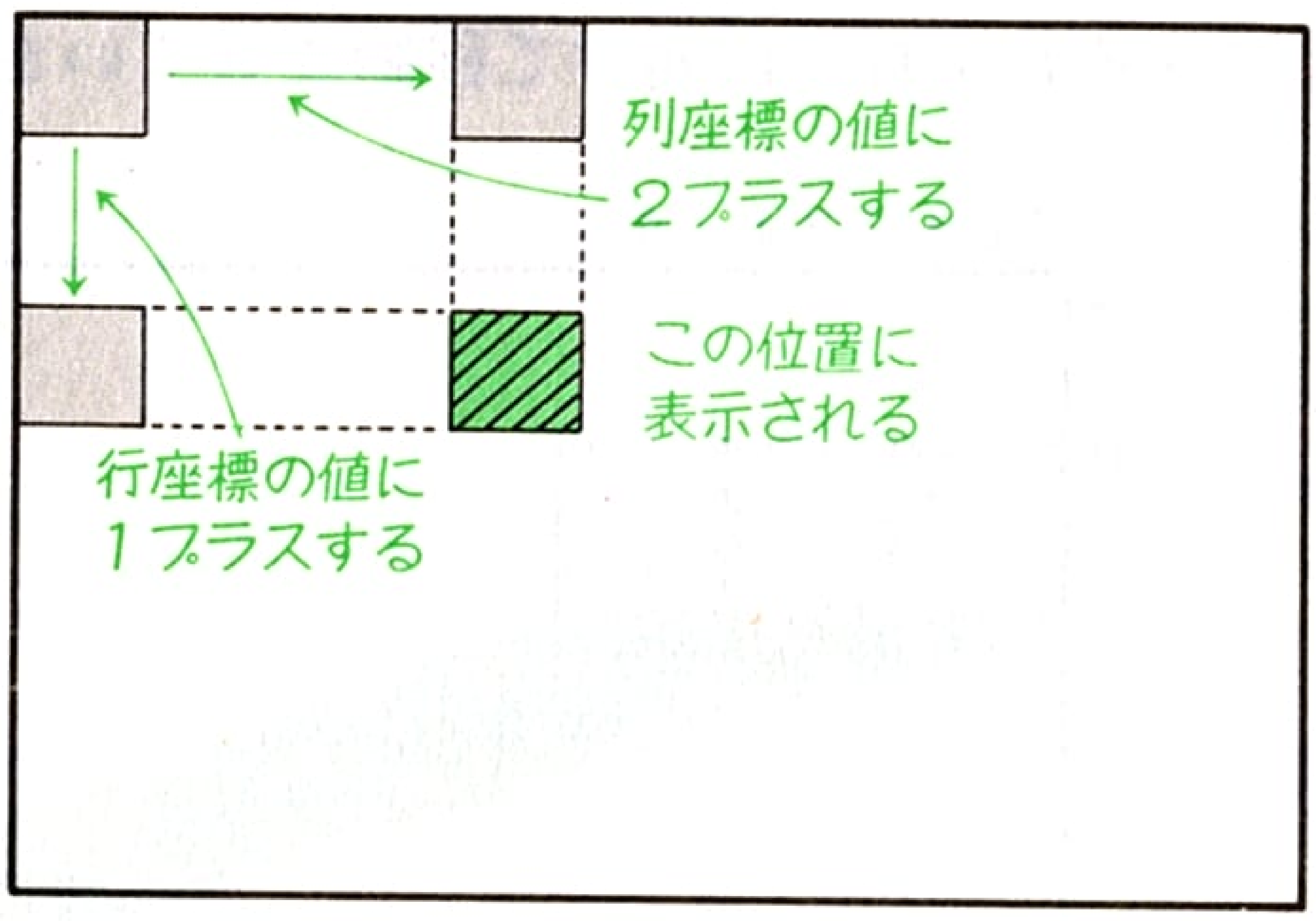
図③



このことを繰り返えしていくと、スプライトは斜めに移動していきます。

1個目のスプライトを移動させる

3個のスプライトを斜めに移動させるには、1個目のスプライトだけを、いま説明した方法で移動させます。2個目と3個目のスプライトを斜めに移動させる方法は異なります。このことについてはあとで説明します。まず、1個目のスプライトの斜めの移動です。



ここでは、右の図に示すように、行座標の値に1をプラスし、列座標の値に2をプラスした位置にスプライトを表示させていくことにします。

したがって、まず、1個目のスプライトの行座標を指定する値を、Aレジスタに取りだしてきます。1個目の行座標を指定する値をAレジスタに取りだすには、行座標を指定する値はE098番地に記憶されていますから、E098番地をカッコで括ってAレジスタのあとにおきます。

```
E033  LD  A, (E098H)
```

← E098番地の行座標を指定する値をAレジスタに取り出す

これでE098番地の行座標を指定する値は、Aレジスタに取りだされます。

行座標を指定する値をAレジスタに取りだしたら、その値に1をプラスします。1をプラスする命令はINC命令です（182頁参照）。

```
E036  INC  A←――Aレジスタの内容に1をプラスする
```

これで、Aレジスタに取りだしてきた値に1がプラスされます。Aレジスタの値に1をプラスしたら、行座標を指定する値を取りだしてきたE098番地に戻します。E098番地にAレジスタの値を戻すには、Aレジスタのまえにカッコで括ってE098番地をおきます。

```
E037  LD  (E098H), A←Aレジスタの値をE098番地に戻す
```

これでAレジスタの値は、E098番地に記憶されます。

つぎに、列座標を指定する値を取りだしてきて、2をプラスします。列座標を指定する値はE099番地に記憶されていますから、E099番地をカッコで括ってAレジスタのあとにおきます。

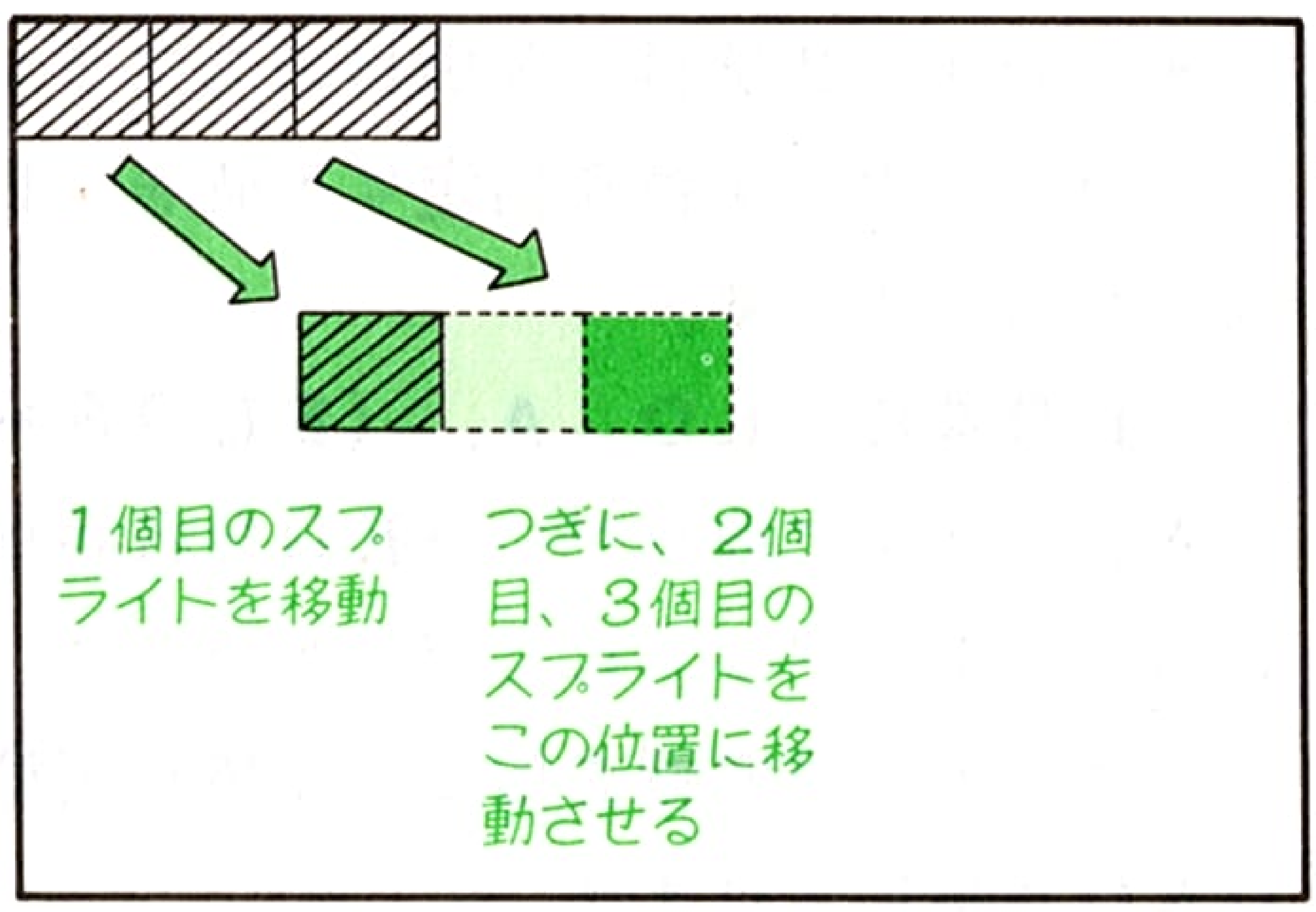
```
E03A  LD  A, (E099H) ←E099番地の列座標の値をAレジスタに取りだす
```

Aレジスタに取りだされてきた値に2をプラスするには、1をプラスするINC命令をふたつづけておきます。

```
E03D  INC  A ｝INC命令を2回実行させて2をプラスする
E03E  INC  A ｝
```

これで、Aレジスタに取りだしてきた値に、1プラス、1プラスで2がプラスされます。Aレジスタの値に2をプラスしたら、列座標を指定する値を取りだしてきたE099番地に戻します。

```
E03F  LD  (E099H), A←Aレジスタの値をE099番地に戻す
```



E098番地とE099番地に値を戻したら、E098番地からの4バイトのデータをスプライト属性テーブルに送ると、右の図に示した位置に、1個目のスプライトが移動して表示されますから、つぎに2個目と3個目のスプライトを、斜め下に移動した1個目のスプライトの横に表示させます。

そのまえに、ウエイトループをおきます。

ウエイトループ

Ｅ０４２番地からＥ０４９番地までの命令は、スプライトの表示を遅らせるためのウエイトループです。

```
E042  LD  A, FFH
E044  LD  B, 0FH
E046  DJNZ  E046H
E048  DEC  A
E049  JR  NZ  E044H
```
ウエイトループ

このウエイトループは、196頁で説明したウエイトループとまったくおなじですから、その頁を参照してください。ただ、196頁の場合は、ＢレジスタにＦＦＨ（10進数で255)を入れていますが、ここでは０ＦＨ（10進数で15）を入れていますから、それだけ速くスプライトの表示が繰り返えされます。つまり、スプライトの移動の速度が速くなるということです。

2個目、3個目のスプライトを移動させる

さて、まえの頁の図に示したように、斜めに移動した１個目のスプライトの横に２個目、３個目のスプライトを表示するには、１個目のスプライトの行座標を指定する値を、Ｅ０９８番地からＡレジスタに取りだしてきます。

```
E04B  LD  A, (E098H)
```
← 1個目のE098番地の1プラスして戻した値をAレジスタに取りだす

Ａレジスタに取りだしたＥ０９８番地の行座標の値は、さきにＩＮＣ命令で１をプラスして戻した値です。この値を２番目と３番目のスプライトの、行座標を指定する値が記憶されているＥ０９Ｃ番地とＥ０Ａ０番地に入れます。

```
LD  (E09CH), A
LD  (E0A0H), A
```
E098番地の値を2個目、3個目の行座標を指定する値が記憶されているメモリ番地に入れる

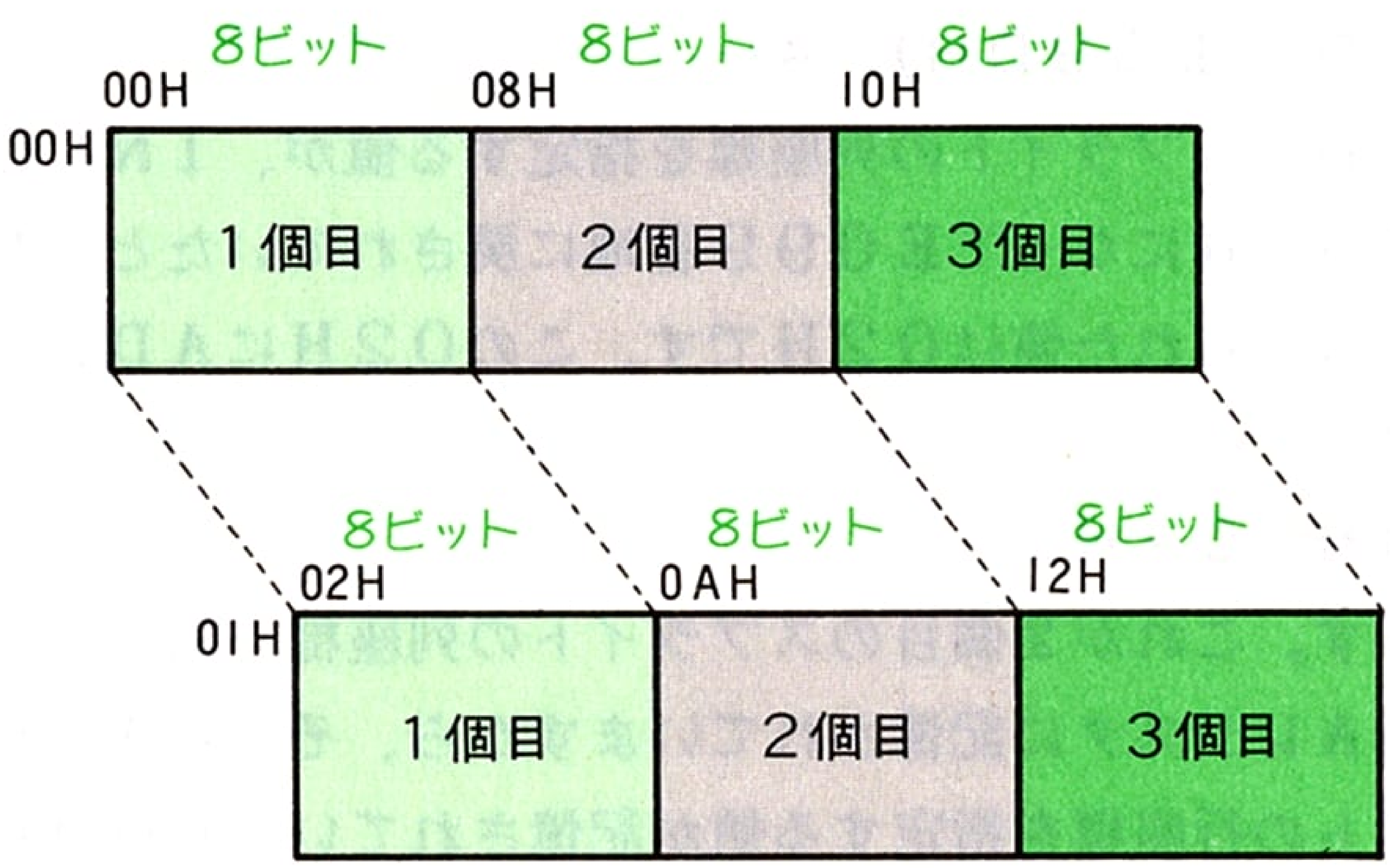
１個目のスプライトの行座標を指定する値を、２個 目と３個目のス

プライトの行座標の値を記憶しているE09C番地とE0A0番地に入れると、つぎに表示される2個目と3個目のスプライトが表示される行座標の位置は、1個目のスプライトとおなじ行座標の位置となります。たとえば、1個目のスプライトの行座標を指定する値が、INC命令で1プラスされて01Hになって E098番地に戻されていたとしたら、その01HをAレジスタに入れて、2個目と3個目の行座標を指定する値が記憶されているE09C番地と、E0A0番地に入れると、1個目から3個目までの行座標を指定する値は、つぎの図に示すようにおなじ値になります。

1個目の行座標	2個目の行座標	3個目の行座標
01H	01H	01H

したがって、この値をスプライト属性テーブルに送ると、つぎに表示される1個目から3個目までのスプライトは、おなじ01Hという行座標の位置に表示されることになるわけです。

このようにして2個目と3個目の、スプライトの行座標を指定する値を、1個目のスプライトの行座標を指定する値とおなじにしたら、つぎは、2個目と3個目のスプライトの列座標を指定する値を変えます。2個目と3個目のスプライトの列座標を指定する値は、つぎの図に示すように、



2個目は1個目より8ビット分右の位置、3個目は2個目より8ビ

ット分右の位置となります。これはスプライトパターンが、横8ビットで作られているからです。

まず、2個目のスプライトの列座標を指定する値を求めます。2個目のスプライトの列座標を指定する値は、1個目のスプライトの列座標を指定する値より8ビット分右の位置ですから、1個目のスプライトの列座標の値が記憶されているE099番地から値を取りだしてきて、Aレジスタに入れます。E099番地に記憶されている値は、INC命令を2回実行して、2プラスして戻した値です。

```
E054  LD  A,(E099H)  ← 1個目のスプライトを表示する列座標の値を取り出してAレジスタに入れる
```

そして、Aレジスタに入れた値に08Hを足します。Aレジスタの値に08Hを足すには、8ビットレジスタに08Hを入れます。ここでは、Dレジスタに08Hを入れることにします。

```
E057  LD  D,08H  ←―― 08HをDレジスタに入れる
```

Dレジスタに08Hを入れたら、足し算の命令であるADD命令で足します(73頁参照)。

```
E059  ADD  A,D  ←―― Aレジスタの値とDレジスタの値を足す
```

これでAレジスタの値に08Hが足されて、2個目のスプライトの列座標を指定する値が求められました。足し算の結果の値はAレジスタに記憶されていますから、その値を2個目のスプライトの列座標を指定する値が記憶されているE09D番地に入れます。

```
E05A  LD  (E09DH),A  ←―― 足し算の結果の値をE09D番地に入れる
```

たとえば、1個目のスプライトの列座標を指定する値が、INC命令を2回実行して02HになってE099番地に戻されていたとしたら、Aレジスタに入れられた値は02Hです。この02HにADD命令で08Hを足すと

02H+08H=0AH

0AHとなります。これが2個目のスプライトの列座標を指定する値です。0AHはAレジスタに記憶されていますから、その0AHを2個目のスプライトの行座標を指定する値が記憶されているE09D番地に入れるわけです。

つぎは、3個目の列座標を指定する値を求めます。3個目のスプライトの列座標を指定する値は、2個目のスプライトの列座標を指定する値より8ビット分右です。2個目のスプライトの列座標を指定する値は、さきの足し算で求められてAレジスタに記憶されていますから、その値に、さらにDレジスタの08Hを足します。

E05D　ADD　A, D ← Aレジスタに記憶されている値に、さらに08Hを足す

たとえば、さきの足し算で、2個目のスプライトの列座標を指定する値は、0AHとなったとします。この0AHは、2個目のスプライトの列座標を指定する値としてE09D番地に入れましたが、Aレジスタにもそのまま記憶されて残っています。したがって、その0AHにさらに08Hを足すと、3個目の列座標を指定する値が求められるわけです。

0AH+08H=12H

足し算の結果は、Aレジスタに記憶されますから、その値を3個目のスプライトの列座標を指定する値が記憶されているE0A1番地に入れます。

E05E　LD　(E0A1H), A ← 足し算の結果をE0A1番地に入れる

これで2個目、3個目のスプライトを表示する値が求められました。たとえば、1個目のスプライトの列座標を指定する値が、INC命令を2回実行して02Hになって、E099番地に戻されていたとすると、2個目のスプライトの列座標を指定する値は、02Hに08Hをプラスしますから、0AHになります。3個目のスプライトの列座標を指定する値は、0AHにさらに08Hをプラスしますから12Hとなります。そして、これらの列座標を指定する値は、2個目、3個目の列座標を指定する値が記憶されているE09D番地、E0A1番地に入れられます。

1個目の列座標	2個目の列座標	3個目の列座標
02H	0AH	12H

この列座標を指定する値を、まえの行座標を指定する値と一緒に、

スプライト属性テーブルに送ると、３個のスプライトが、それまで表示されていた位置より斜め下に移動して、表示されることになります。

DJNZ命令で繰り返えす

あとはＤＪＮＺ命令で、これまで説明してきたことを、プログラムの最初のほうでＢレジスタに入れた７０Ｈ（10進数で112）回繰り返えします。

ＤＪＮＺ命令で繰り返えし処理するために戻るとき、Ｂレジスタの値から１を引いて戻ります。

DJNZ命令＝Bレジスタから１を引いて戻る

Ｂレジスタに入れた値は、そのあとでＢレジスタを使う関係から、ＰＵＳＨ命令でスタックに退避させました。スタックに退避させたままでは引くことができませんから、ＰＯＰ命令でスタックからＢレジスタに取りだしてきます。

```
E061  POP  BC
```
←Bレジスタに70Hを取りだす

これでＤＪＮＺ命令で繰り返えし処理するために戻るとき１を引くことができますから、そのあとにＤＪＮＺ命令をおきます。

```
E062  DJNZ  E00EH
```
←DJNZ命令で繰り返えす

ＤＪＮＺ命令で繰り返えしのために戻る最初の命令は、Ｅ００Ｅ番地のＰＵＳＨ　ＢＣです。

なぜ、スプライト属性テーブルにデータを送る最初のＥ００Ｆ番地のＬＤ　ＨＬ，Ｅ０９８Ｈの命令に戻さないのかというと、Ｅ００Ｆ番地の命令のあとで、Ｂレジスタに新たにデータを入れて使っているからです。そのため、ＤＪＮＺ命令で戻すさきをＥ００Ｅ番地のＰＵＳＨ　ＢＣ命令にして、ＤＪＮＺ命令で戻るたびにＢレジスタの値をスタックに退避させて、繰り返えしの回数が破壊されてわからなくならないようにするのです。

最後にＲＥＴ命令をおくと、３個のスプライトを画面の左上から右下に向けて、移動するプログラムはおわりです。

絶対アドレッシングと相対アドレッシング

アドレッシングとは

アドレッシングにはいろいろありますが、この本で取りあげたプログラムでは、ＪＲ命令やＤＪＮＺ命令を使っていますので、絶対アドレッシングと相対アドレッシングのふたつについて、説明しておくことにします。

アドレッシングとは、メモリ番地を指定することです。わかりやすいように、ＭＳＸベーシックのプログラムを使って説明します。

```
10 LOCATE X, 10
20 PRINT "●"
30 X=X+1
40 IF X=20 THEN 60
50 GOTO 10
60 END
```

このプログラムは、20個の●を画面の左端から右に向かって表示するものです。画面の左端から右に向かって20個の●を表示させていくには、行番号10から30までのステートメントを20回繰り返えす必要がありますから、行番号50のＧＯＴＯ文で行番号10にジャンプさせて戻しています。●が20個表示されると行番号40のＩＦ文で判断されて、ＴＨＥＮ　60で行番号60のＥＮＤ文にジャンプしておわります。

行番号50のＧＯＴＯ文でわざわざ行番号10に戻し、ＩＦ文で判断させておわりとするなどということをしなくても、ＦＯＲ～ＮＥＸＴ文を使えば、もっとすっきりしたプログラムになりますが、説明の都合上、このような方法を取ったわけです。

このようにMSXベーシックの場合は、GOTOのあとに行番号を指定すると、その行番号にジャンプして、その行番号のステートメントから実行していきます。THEN文の場合も行番号を指定すると、その行番号にジャンプして、その行番号のステートメントから実行していきます。マシン語の場合は、このようなわけにはいきません。なぜなら、マシン語には行番号がないからです。

そこでマシン語の場合は、ジャンプして実行させる命令が記憶されているメモリ番地を指定します。このように、ジャンプすべきメモリ番地を指定することを**アドレッシング**といいます。

もちろん、ジャンプするさきのメモリ番地の指定だけをアドレッシングというわけではありません。メモリに記憶されているデータを取りだすときも、メモリ番地の指定をしますから、これもアドレッシングです。

さて、アドレッシングがメモリ番地の指定であることがわかったところで、絶対アドレッシングと相対アドレッシングです。

絶対アドレッシングは、命令に直接メモリ番地を書く

つぎのアセンブリ言語の**JP（ジャンプ）命令**と、**JR（ジャンプ・リラティブ）命令**は、ふたつともおなじようにE00C番地にジャンプする命令です。アセンブリ言語では、両方ともメモリ番地の指定はE00CHですが、

```
JP  E00CH ←――― E00C番地にジャンプ
JR  E00CH ←――― E00C番地にジャンプ
```

これをマシン語になおしてみると、アドレッシング、つまり、メモリ番地の指定の仕方が違うことがわかります。JP　E00Cをマシン語になおすには、142頁のZ－80命令表のJP　×，nnの欄とUN COND（無条件）の項とが交わったところをみます。JR　E00CHをマシン語になおすには、JR　×，eとUN　COND（無条件）の項とが交わったところをみます。

```
C3  nn ←――― JP  E00CHのマシン語
```

18　e－2←――JR　E00CHのマシン語

ｎｎとｅ－２のところにメモリ番地を指定しますが、ＪＰ命令はｎｎで、ＪＲ命令はｅ－２となっていますから、メモリ番地の指定の仕方が違うということになります。

ｎｎは、２バイトの数値を示しますから、ｎｎには、２バイトのメモリ番地を直接指定することになります。

ｎｎ＝２バイトのメモリ番地を直接指定する

ｅ－２のメモリ番地の指定の仕方についてはあとで説明します。

したがって、ＪＰ　Ｅ００ＣＨのＥ００ＣＨはｎｎに直接指定しますから、

JP　E00CH

↓

JP　nn←――2バイトのメモリ番地を直接指定する

ＪＰ　Ｅ００ＣＨのマシン語は、

C3　0C　E0

となります。Ｅ００ＣＨという２バイトの数値をマシン語になおすときは、上位２桁をあとに、下位２桁をさきにします。

このように、アセンブリ言語をマシン語になおすとき、マシン語の命令に直接メモリ番地を指定することを、**絶対アドレッシング**といいます。

絶対アドレッシング＝マシン語の命令に直接メモリ番地を指定



また、絶対アドレッシングでメモリ番地を指定するジャンプ命令を**絶対ジャンプ命令**といいます。

絶対ジャンプ命令＝絶対アドレッシングでメモリ番地を指定

つぎは、e－2のメモリ番地の指定の仕方です。e－2のメモリ番地の指定の仕方には、プログラムカウンタの働きがかかわってくるので、そのまえにプログラムカウンタの働きについて説明します。

プログラムカウンタの働き

プログラムカウンタは、その頭文字を取って、ＰＣと書き表わします。ＰＣは、ＣＰＵのなかにある16ビットのレジスタです。ＰＣは16ビットのレジスタですから、16ビットの数字を記憶することもできますが、それとともに、数をカウントする働きをもっています。

このように、ＰＣが数値を記憶することができて、しかも数をカウントする働きをもっているのは、プログラムの命令が記憶されているメモリ番地をカウントして、つぎの命令が記憶されているメモリ番地をＣＰＵに指示することが、ＰＣの働きだからです。つぎに実際にプログラムを使って、ＰＣの働きを説明することにします。

メモリ番地	マシン語	アセンブリ語
E000	06 14	LD B, 14H
E002	3E 0D	LD A, 0DH
E004	32 DC F3	LD (F3DCH), A
E007	3E 01	LD A, 01H
E009	32 DD F3	LD (F3DDH), A
E00C	3E 85	LD A, 85H
E00E	CD A2 00	CALL 00A2H
E011	05	DEC B
E012	20 F8	JR NZ E00CH
E014	C9	RET

まず、マシン語モニタをＲＵＮさせ、このプログラムを実行させます。実行させるコマンドはＧコマンドです。そして、プログラムが記憶されている最初のメモリ番地を入力します。

メモリバンチ　ノ　ニュウリョク？　E000

Ｅ０００番地を入力して、RETキーを叩くと、ＰＣには、

PC　⬅　E000

Ｅ０００番地がセットされて、ＰＣはＥ０００番地を指示することになりますから、ＣＰＵは、ＰＣが指示しているＥ０００番地からの命令の実行に取りかかります。Ｅ０００番地からの命令は、

06　14

ですから、ＣＰＵがＥ０００番地からの命令の実行にかかると、ＰＣは、つぎの命令が記憶されているＥ００２番地を指示します。

PC　➡　E002

ＣＰＵはＥ０００番地からの命令を実行し終わると、つぎのＰＣが指示しているＥ００２番地からの命令の実行に取りかかります。Ｅ００２番地からの命令は、

3E　0D

ですから、ＣＰＵがＥ００２番地からの命令の実行に取りかかると、ＰＣは、つぎの命令が記憶されているＥ００４番地を指示します。

PC　➡　E004

ＣＰＵは、Ｅ００２番地の命令を実行し終わると、ＰＣが指示しているＥ００４番地からの命令の実行に取りかかります。Ｅ００４番地からの命令は、

32　DC　F3

ですから、ＣＰＵがＥ００４番地からの命令の実行に取りかかると、ＰＣは、つぎの命令が記憶されているＥ００７番地を指示します。

PC　➡　E007

このようにＰＣは、ＣＰＵがつぎに実行する命令が記憶されているメモリ番地を指示します。ＣＰＵは、ＰＣの指示しているメモリ番地にしたがって、そのメモリ番地から命令を取りだして処理します。

CPU=PCが指示するメモリ番地の命令を実行

さて、プログラムのＥ０１２番地、つまり、下から２行目をみてください。ＪＲ命令になっています。最初に取りあげたのは、

JR　E00CH

でした。この形は、無条件ジャンプ命令です。この場合は、

JR　NZ　E00CH

で、ＮＺ（ゼロフラグが０のとき）という条件がついていますから、条件つきジャンプ命令となります。つまり、ゼロフラグが０のときＥ００Ｃ番地にジャンプするということです。ゼロフラグが１になったら、ＮＺの条件は成立しませんから、Ｅ００Ｃ番地にジャンプしないで、つぎのメモリ番地の命令の実行にかかります。

ところで、ＪＲ　Ｅ００ＣＨのマシン語は、

18　e−2

で、ｅ－２にジャンプするさきのメモリ番地を指定するということでした。ＪＲ　ＮＺ　Ｅ００ＣＨのマシン語は､142頁のＺ－80命令表のＪＲ　×，ｅの欄とＮＺの項の交わったところをみます。

20　e−2

で、メモリ番地の指定のところはおなじｅ－２ですから、ジャンプするさきのメモリ番地はおなじメモリ番地の指定の仕方となります。

JR　E00CH	JR　NZ　E00CH
↓	↓
18　e−2	20　e−2

e−2のメモリ番地の指定は相対アドレッシング

さきに説明したように、ｎｎの場合は直接２バイトのメモリ番地を指定する絶対アドレッシングというメモリ番地の指定の仕方でしたが、ｅ－２はｎｎと違い、マシン語の命令に直接メモリ番地を書きません。

226頁に示したプログラムのＪＲ　ＮＺ　Ｅ００ＣＨのマシン語を

みると、

```
JR  NZ  E00CH
 ↓
20  F8
```

となっています。20がJR　NZのマシン語で、F8がE00CHです。ところで、F8は16進数2桁です。したがって、1バイトとなります。

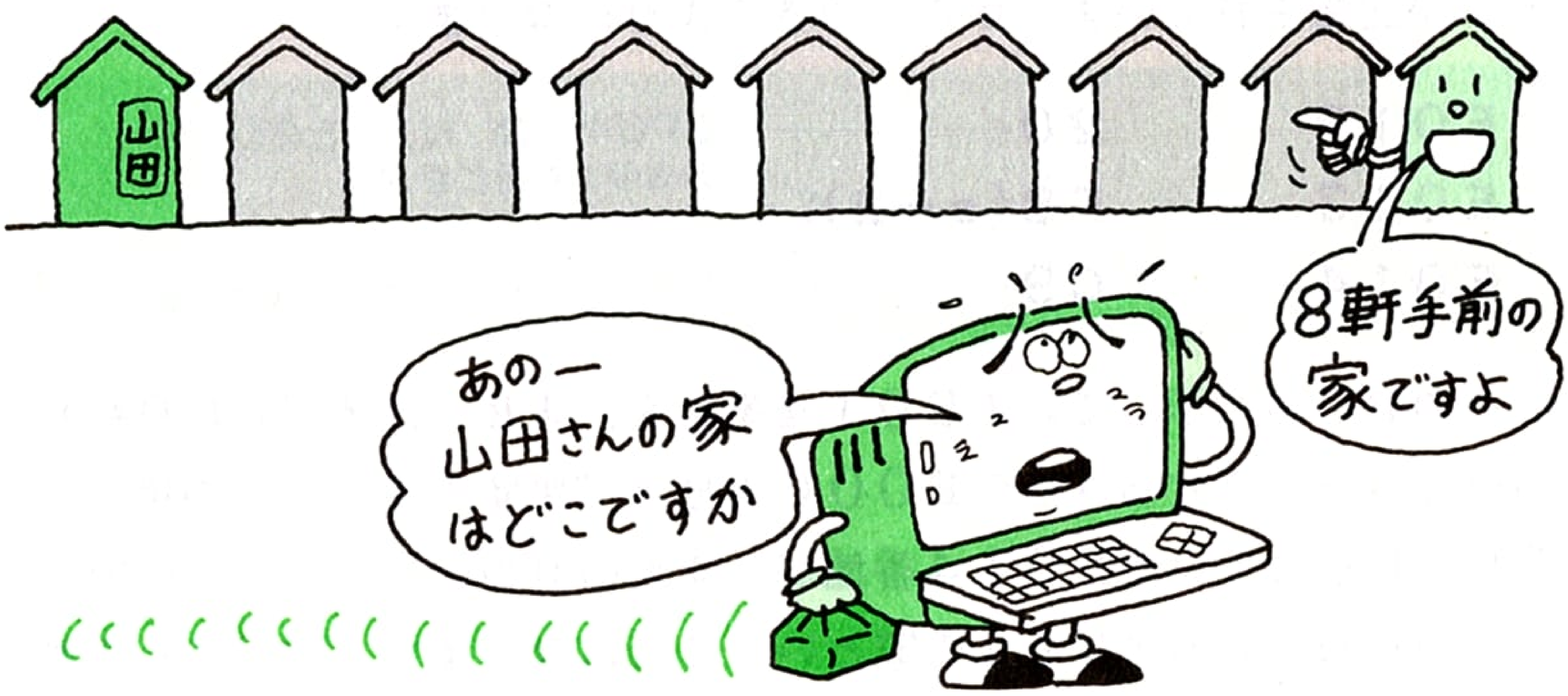
F8＝16進数2桁なので1バイト

このようにe－2の場合、E00C番地というような2バイトのメモリ番地は、F8というような1バイトで指定するということに決まっています。

e－2＝1バイトでメモリ番地を指定する

では、E00CHという2バイトのメモリ番地を、どうすればF8という1バイトで指定することができるのでしょうか。

絶対アドレッシングでは、直接マシン語の命令にE00C番地を指定して、E00C番地へジャンプせよでしたが、この場合は、あるメモリ番地を基準にして、いくつ手前のメモリ番地まで戻れ、というよ



うなメモリ番地の指定の仕方をするのです。たとえば、探している家を通り越して、××さんの家で○○さんの家はどこかと尋ねたとします。尋ねられた人が、その人がどちらから来たかを知っていると、○○さんの家は8軒手前の家ですよ、と答えることがあります。つまり8軒戻れということです。この答え方のようなメモリ番地の指定の仕方をするわけです。

このように、あるメモリ番地を基準にして、8軒先にまで戻れというようなメモリ番地の指定の仕方を、**相対アドレッシング**といいます。

相対アドレッシングのメモリ番地の指定の仕方

では、相対アドレッシングでは、メモリ番地をどのように指定するのか、実際に行ってみることにします。

ここで必要になるのが、まえに説明したＰＣ(プログラムカウンタ)の働きです。ＣＰＵは、ＰＣが指示しているメモリ番地から命令を取りだして実行します。この場合は、相対アドレッシングでメモリ番地を指定するＪＲ　ＮＺ　Ｅ００ＣＨ命令は、226頁のプログラムの下から2行目のＥ０１２番地にありますから、ＰＣがＥ０１２番地を指示するところからはじめます。ＣＰＵが、ＰＣが指示しているＥ０１２番地の命令の実行に取りかかると、ＰＣはつぎの命令が記憶されているＥ０１４番地を指示します。マシン語は、ひとつのメモリに1バイトずつ記憶されていますから、つぎのようになります。

E012	20	←CPUがJR NZ E00CHの命令の実行にかかる
E013	F8(e−2)	
E014	C9	←PCはこのメモリ番地を指示する

このＰＣが指示しているＥ０１４番地を、ＪＲ　ＮＺ　Ｅ００ＣＨ命令でジャンプするさきのＥ００Ｃ番地まで戻すのです。この場合、ＰＣが指示しているＥ０１４番地が、まえの頁の図の○○さんの家がどこか尋ねている××さんの家ということになります。○○さんの家は、ＪＲ　ＮＺ　Ｅ００ＣＨでジャンプするさきのＥ００Ｃ番地です。

つまり、マシン語で３Ｅです。

```
E00C  3E←――――――〇〇さんの家
E00D  85       (JR NZ E000Hで戻るところ)
E00E  CD
E00F  A2
E010  00
E011  05
E012  20
E013  F8 (e−2)
E014  C9←――――――××さんの家
               (PCが指示しているメモリ番地)
```

さて、Ｅ００Ｃ番地はＥ０１４番地からさかのぼって何軒さき、つまり何バイトさきにあるでしょうか。この数え方は、つぎのようにして行います。

```
E00C  3E←――――――――――――−8すると、このメモリ番地
E00D  85←――――――――――――−7すると、このメモリ番地
E00E  CD←――――――――――――−6すると、このメモリ番地
E00F  A2←――――――――――――−5すると、このメモリ番地
E010  00←――――――――――――−4すると、このメモリ番地
E011  05←――――――――――――−3すると、このメモリ番地
E012  20←――――――――――――−2すると、このメモリ番地
E013  F8(e−2)←―――――――−1すると、このメモリ番地
E014  C9←――――――――――――PCが指示しているメモリ番地
```

つまり、ＰＣが指示しているメモリ番地から−８すると、Ｅ００Ｃ番地となりますから、ＪＲ　ＮＺ　Ｅ００ＣＨ命令の場合は、

20　e−2

のｅ−２のところに、

e−2=−8

−８をあてはめればよいということになります。ただし、−８は10

進数ですから、そのままあてはめることはできません。－８を16進数になおします。マイナスの符号のついた10進数を16進数になおすには、233頁に示した符号つき16進数一覧表を使ってなおします。

－8＝F8H

10進数の－８は、16進数でＦ８Ｈですから、ｅ－２には、

e－2＝F8

Ｆ８Ｈをあてはめることになります。したがって、

JR NZ E00CH

↓

20 F8

となるのです。これで、１バイトでメモリ番地の指定ができたわけです。

ＪＲ　Ｅ００ＣＨの場合も、

18 e－2

ｅ－２のメモリ番地の指定は、このようにしてメモリ番地の指定を行います。

繰り返えしの命令ＤＪＮＺ命令の場合も、

10 e－2

メモリ番地の指定のところはｅ－２ですから、いま説明したようにしてメモリ番地の指定を行います。

以上が、相対アドレッシングのメモリ番地の指定の仕方です。



符号つき16進数

上位＼下位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	-128	-127	126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113
9	-112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

フラグレジスタとは

フラグレジスタは、特殊な働きをするレジスタ

150頁で示したレジスタの図のなかで、Aレジスタの横にFと書かれたレジスタがありました。このFが**フラグレジスタ**です。フラグレジスタは、ふつう**フラグ**と呼んでいます。

このフラグレジスタは、A、B、C、D、E、H、Lなどのレジスタと違って、データを出し入れすることができません。というのは、フラグレジスタは特殊な働きをするレジスタだからです。

個々のビットには、固有の名前がついている

フラグレジスタは、A、B、C、D、E、H、Lなどのレジスタとおなじように8ビットレジスタですから、AからLまでのレジスタとおなじように、8つの記憶場所をもっています。この8つの記憶場所は0か1の1ビットを記憶するもので、つぎの図に示すように、ひとつひとつの記憶場所には右から順に、ビット0からビット7までのビット番号がついています。そしてまた、ビット番号ごとに固有の名前がつけられています。固有の名前は記号で表わされます。

フラグレジスタ

7	6	5	4	3	2	1	0	ビット番号
S	Z	×	H	×	P/V	N	C	記号
↑サインフラグ	↑ゼロフラグ		↑ハーフ・キャリーフラグ		↑パリティ／オーバフローフラグ	↑加／減算フラグ	↑キャリーフラグ	

そして、**Cはキャリーフラグ、Zはゼロフラグ**というように呼んでいます。ビット番号3と5は×になっていますが、これは未定義ということです。つまり、使っていないということですから、フラグとして働くのは、

S　Z　H　P/V　N　C

の6つです。この6つのフラグの働きを、最初から全部おぼえる必要はありません。ふつう使うフラグは、ゼロフラグ（Z）とキャリーフラグ（C）の2つですから、ゼロフラグとキャリーフラグの2つの働きをしっかりおぼえて、そのあと、ほかのフラグの働きをおぼえていけばよいでしょう。

フラグは、演算の結果で1か0になる

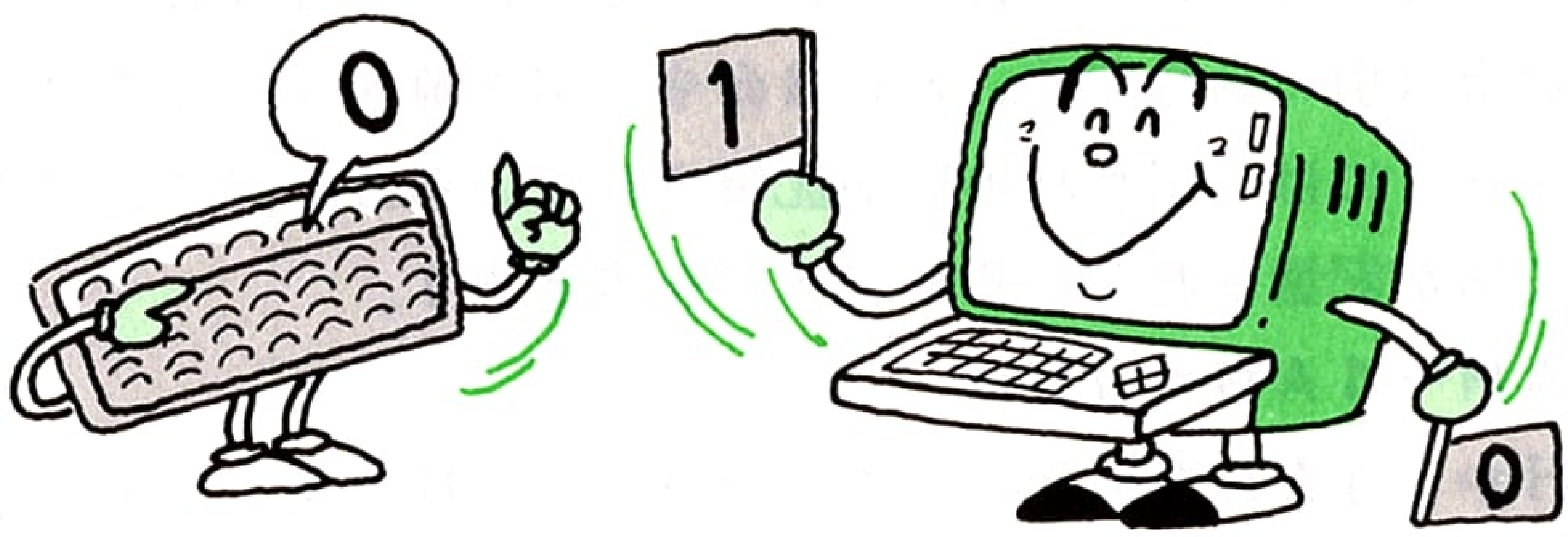
さて、この6つのフラグは、さきに説明したように、それぞれ1か0の1ビットを記憶することができますから、1になったり、0になったりします。

フラグ＝1か0の値になる

では、どんなときに1になったり、0になったりするのでしょうか。

それぞれのフラグが1になったり、0になったりするのは、それぞれのフラグによって決まっていますが、どのフラグも演算の結果によって、1になったり、0になったりします。

フラグの値＝演算の結果で1か0になる



マシン語のプログラムでよく使われるフラグは、ゼロフラグとキャリーフラグの２つですから、ここではこの２つのフラグについて、どんなとき１になり、どんなとき０になるのか、説明することにします。

ゼロフラグは、演算の結果が０のとき１、０でないとき０

最初はゼロフラグです。ゼロフラグは、演算の結果が０になったとき１になります。演算の結果が０にならないときは０になります。

演算の結果が０のとき＝ゼロフラグは１になる
演算の結果が０でないとき＝ゼロフラグは０になる

つぎのプログラムで、具体的に説明することにします。

```
LD  A, 25H←――25HをAレジスタに入れる
ADD  A, 25H←――Aレジスタの値に25Hを足す
SUB  4AH←――Aレジスタの値から4AHを引く
```

１行目のＬＤ　Ａ，２５Ｈで、Ａレジスタに２５Ｈ（10進数で37）を入れます。２行目のＡＤＤ　Ａ，２５Ｈで、１行目でＡレジスタに入れた２５Ｈに２５Ｈを足します。ＡＤＤは足す命令です。

２５Ｈ＋２５Ｈ＝４ＡＨ

足し算の結果は４ＡＨ（10進数で74）で、０ではありませんから、ゼロフラグは０になります。

足し算の結果４ＡＨ＝ゼロフラグは０になる

足し算の結果の４ＡＨは、Ａレジスタに記憶されていますから、３行目のＳＵＢ　４ＡＨで、Ａレジスタの４ＡＨから４ＡＨを引きます。ＳＵＢ命令は引く命令です。ＳＵＢ命令で引き算をするときは、ＳＵＢ命令のあとには、引かれる値が記憶されているＡレジスタは書かずに、引く値が記憶されているレジスタを書きます。

４ＡＨ−４ＡＨ＝００Ｈ

４ＡＨから４ＡＨを引くと、その結果は００Ｈです。つまり10進数で０です。演算の結果が０であると、ゼロフラグは１になります。

演算の結果が００Ｈ＝ゼロフラグは１になる

これがゼロフラグの働きです。このように変化するゼロフラグをどのように使うのか、ということについては、あとで説明します。

なお、フラグが1になったり、0になったりすることを、**セット（1になる）、リセット（0になる）**といいます。また、フラグが1になることを、**フラグが立つ**などともいいます。

キャリーフラグが、1あるいは0になるときは

つぎは、キャリーフラグです。キャリーフラグは、234頁の表ではCという記号で示しましたが、CYという記号でも書き表わします。どちらを使ってもかまいません。

さて、キャリーフラグはゼロフラグよりも、少し複雑です。キャリーフラグが1になったり、0になったりするときは、足し算と引き算の2つにわけて考えることができます。

足し算で、桁あふれをおこしたとき1

ある値とある値を足したとき、桁上がり、つまり桁あふれがおきることがあります。このように桁あふれがおきたとき、キャリーフラグは1になります。桁あふれについては、あとで例をあげて説明します。

足し算の結果、桁あふれがおきたとき＝キャリーフラグは1

桁あふれがおきないとき、キャリーフラグは0になります。

足し算の結果、桁あふれがおきないとき＝キャリーフラグは0

ではつぎに、キャリーフラグが1になるとき、また0になるときを例を使って具体的に説明することにします。

▶足し算で桁あふれがおきた、おきない

```
LD   A, 64H ←――Aレジスタに64Hを入れる
ADD  A, 9CH ←――Aレジスタの値に9CHを足す
SUB  9CH ←――――Aレジスタの値から9CHを引く
ADD  A, 32H ←――Aレジスタの値に32Hを足す
```

１行目のＬＤ　Ａ，６４Ｈで、Ａレジスタに６４Ｈ（10進数で100）を入れます。２行目のＡＤＤ　Ａ，９ＣＨで、１行目でＡレジスタに入れた６４Ｈに９ＣＨ（10進数で156）を足します。

５４Ｈ＋９ＣＨ＝１００Ｈ

６４Ｈに９ＣＨを足すと、１００Ｈ（10進数で256）になります。

さて、１００Ｈは何ビットの数でしょうか。８ビットの数は16進数２桁で表わすことができる数ですから、００Ｈ（10進数で０）からＦＦＨ（10進数で255）までの数です。したがって、１００Ｈは16ビットの数ということになります。

１００Ｈ＝16ビットの数

この足し算の結果の１００Ｈは、Ａレジスタに記憶されて残りますが、Ａレジスタは８ビットのレジスタです。

Ａレジスタ＝８ビットレジスタ

８ビットのＡレジスタは、16ビットの１００Ｈという数を記憶することができません。このように８ビットの数６４Ｈと、８ビットの数９ＣＨを足して、その結果が１００Ｈという16ビットの数になってしまって、８ビットのＡレジスタに記憶することができなくなってしまうことを、**桁あふれをおこした**といいます。

桁あふれをおこす＝レジスタに記憶することができなくなる

このように桁あふれがおきたとき、キャリーフラグは１になります。

桁あふれがおきたとき＝キャリーフラグは１

では、桁あふれがおきないときです。３行目のＳＵＢ　９ＣＨで、Ａレジスタの値から９ＣＨを引きます。ＳＵＢ命令は引く命令です。

１００Ｈ－９ＣＨ＝６４Ｈ

Ａレジスタに記憶されている１００Ｈから９ＣＨを引くと、６４ＨがＡレジスタに残ります。４行目のＡＤＤ　Ａ，３２Ｈで、Ａレジスタの６４Ｈ（10進数で100）に３２Ｈ（10進数で50）を足します。

６４Ｈ＋３２Ｈ＝９６Ｈ

足し算の結果は９６Ｈ（10進数で150）です。９６Ｈは、16進数２桁

ですから、8ビットの数です。

９６H＝8ビットの数

したがって、まえの場合と違って、8ビットのAレジスタに記憶することができます。このように8ビットの数６４Hと、8ビットの数３２Hを足した結果が、Aレジスタに記憶することができる8ビットの数であるとき、桁あふれをおこさないということになります。

桁あふれをおこさない＝レジスタに記憶することができる

このように桁あふれがおきないときは、キャリーフラグは0になります。

桁あふれがおきないとき＝キャリーフラグは0

引き算で、桁下がりがおきたときキャリーフラグは1

つぎは、引き算の場合です。ある値からある値を引いたとき、桁下がり、つまり引き算の結果がマイナスの値になることがあります。このようにマイナスの値になったとき、キャリーフラグは1になります。

引き算の結果がマイナスの値＝キャリーフラグは1

ある値からある値を引いたとき、その結果の値がプラスの値か、0の値であるときは、桁下がりがおきないときです。このように桁下がりがおきないときは、キャリーフラグは0になります。

引き算の結果がプラスか0の値＝キャリーフラグは0

▶引き算で桁下がりがおきた、おきない

```
LD  A, 64H ←———— Aレジスタに64Hを入れる
SUB  46H ←———— Aレジスタの値から46Hを引く
SUB  1FH ←———— Aレジスタの値から1FHを引く
```

1行目のLD　A，６４Hで、Aレジスタに６４H（10進数で100）を入れます。2行目のSUB　４６Hで、1行目でAレジスタに入れた６４Hから４６H（10進数で70）を引きます。

６４H−４６H＝１EH

引き算の結果は１ＥＨ（10進数で30）ですから、引き算の結果はプラスの値です。このように、引き算の結果がプラスの値か、０の値であるときは、桁下がりがおきていません。

桁下がりがおきない＝プラスの値か、０の値

このように桁下がりがおきていないときは、キャリーフラグは０になります。

桁下がりがおきないとき＝キャリーフラグは0

２行目の引き算の結果の１ＥＨは、Ａレジスタに記憶されていますから、３行目のＳＵＢ　１ＦＨで、１ＥＨから１ＦＨ（10進数で31）を引きます。

１ＥＨ－１ＦＨ＝ＦＦＨ

１ＥＨ（10進数で30）から１ＦＨ（10進数で31）を引くと、引き算の結果はＦＦＨ（10進数で－１）になります。このように引き算の結果がマイナスの値になると、桁下がりがおきたということになります。

桁下がりがおきた＝マイナスの値

引き算の結果がマイナスの値になって、桁下がりがおきたときはキャリーフラグは１になります。

桁下がりがおきたとき＝キャリーフラグは1

ところで16進数のＦＦＨは、10進数で255を表わしますが、このように10進数で－１をも表わします。

ＦＦＨ＝10進数で255　　　**ＦＦＨ＝10進数で－１**

ＦＦＨが－１を表わしているときは、キャリーフラグが１になっているので、ＭＳＸパソコンは、マイナスの値であることを判断しているのです。

ちなみに、16進数のＦＥＨは、10進数で254を表わしますが、また、10進数で－２を表わします。

ＦＥＨ＝10進数で254　　　**ＦＥＨ＝10進数で－２**

16進数のＦＯＨは、10進数で240を表わしますが、また、10進数で

－16をも表わします。

　　ＦＯＨ＝10進数で240　　　　ＦＯＨ＝10進数で－16

　ＦＥＨやＦＯＨなどがプラスの値であるが、マイナスの値であるかは、キャリーフラグが０であるか、１であるかで判断しています。

　　マイナスの値かプラスの値か＝キャリーフラグが１か０で判断

　そのほかの10進数の－３や－４などの値が、16進数でどのように表わされるのかということについては、233頁に示した符号つき16進数の表を参照してください。

マシン語のプログラムでのフラグの使い方

　これまで、ゼロフラグとキャリーフラグの働きについて説明してきました。フラグは、このように演算が行われるたびに、演算の結果によって１になったり、０になったりしています。

　このフラグの働きを、実際にマシン語のプログラムでどのように使うのか、ここでは、最もよく使うゼロフラグを例に取りあげます。

　ゼロフラグの働きをマシン語のプログラムで使うのは、ほとんどの場合が条件分岐（じょうけんぶんき）のときです。条件分岐は、ある条件にもとづいてプログラムの流れを変えることです。条件分岐をＭＳＸベーシックで書くと、たとえば、つぎのようになります。

```
IF Z=0 THEN 9000
IF Z=1 THEN 9000
```

　ゼロフラグは、演算の結果によって０か１になりますが、その０か１が、うえに示したＭＳＸベーシックのＺ＝０の０かＺ＝１の１にあたります。つまり、条件を判断する材料です。

　では、うえに示したＭＳＸベーシックをアセンブリ言語で書き表わしてみます。最初のＩＦ　Ｚ＝０　ＴＨＥＮ　９０００のＺ＝０を、ゼロフラグが０のときという条件にすると、つぎのようになります。

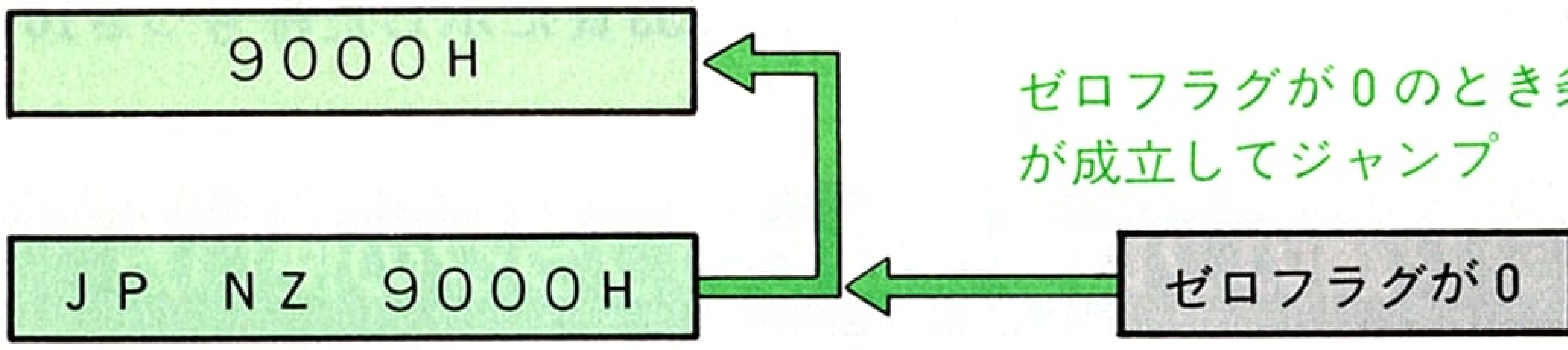
```
JP NZ 9000
```
←ゼロフラグが0のとき9000番地にジャンプ

　ＪＰ命令のＪＰは、英語のjump（ジャンプ）の略記号です。英語のjumpは「飛ぶ」という意味です。

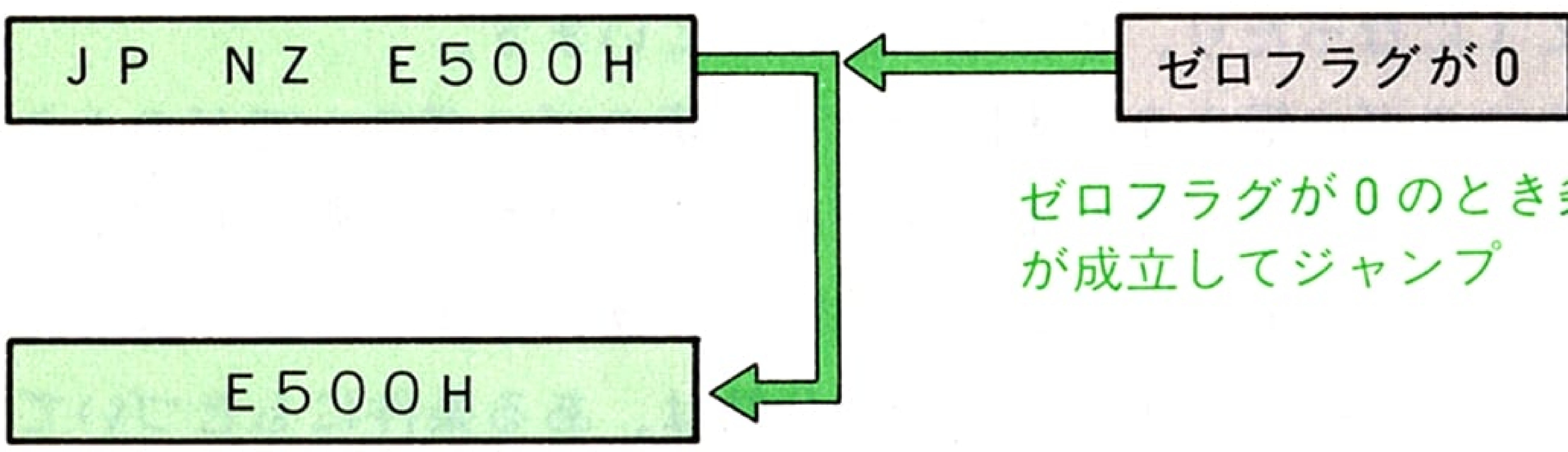
JP　9000H ← 9000番地にジャンプ

とすると、無条件に9000番地にジャンプします。

NZは、non zero（ノン・ゼロ）ということで、ゼロフラグが0のときという条件です。したがって、ゼロフラグが0のとき条件が成立してジャンプします。ジャンプするさきは、NZのあとのメモリ番地です。この例の場合は9000番地となります。



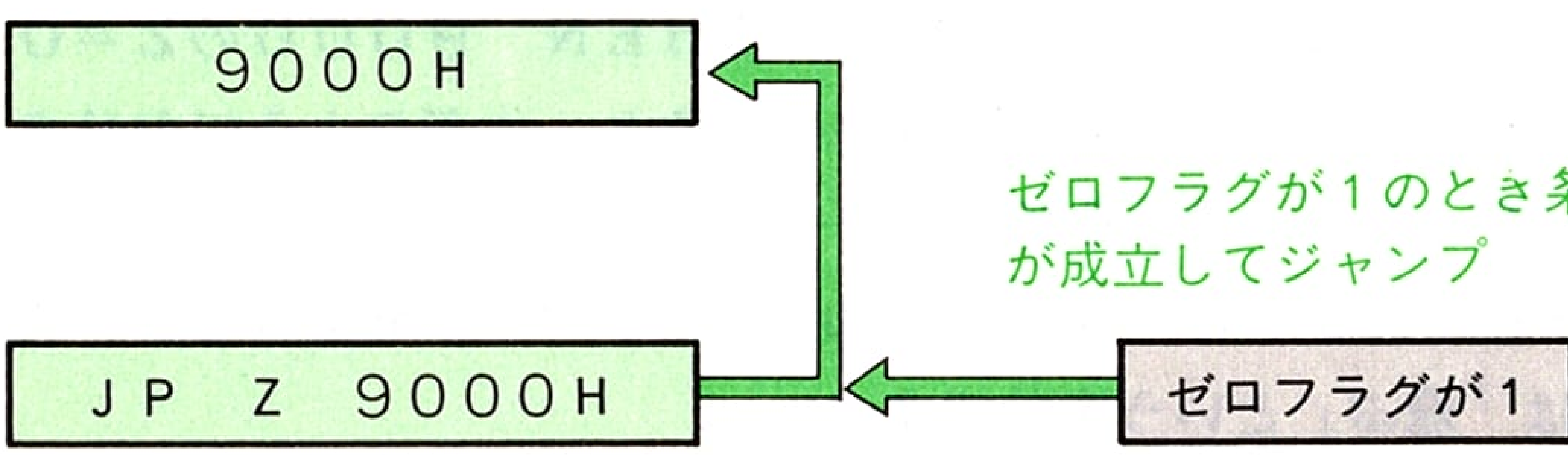
ゼロフラグが0のときジャンプするさきは、JP　NZ命令のまえのメモリ番地であっても、あとのメモリ番地でもかまいません。



2番目のIF　Z＝1　THEN　9000のZ＝1を、ゼロフラグが1のときとして、これをアセンブリ言語で書き表わすと、つぎのようになります。

JP　Z　9000H

JP命令は、さきに説明したとおりです。Zはzero（ゼロ）ということで、ゼロフラグが1のときという条件です。したがって、ゼロフラグが1のとき、条件が成立してジャンプします。ジャンプするさきは、Zのあとにあるメモリ番地です。



ゼロフラグが１のときジャンプするさきのメモリ番地は、ＪＰ　Ｚ命令があるメモリ番地より、まえのメモリ番地でも、あとのメモリ番地であってもかまいません。なお、ＪＰ命令は、さきに説明した、

```
JP  9000H ←無条件ジャンプ命令
```

とすると、ＪＰ命令のあとに示されたメモリ番地に無条件にジャンプするので、無条件ジャンプ命令といいますが、ＪＰ命令のあとに、ＪＰ　ＮＺ、ＪＰ　Ｚという条件をおくと、条件によってジャンプするので、条件つきジャンプ命令といいます。

```
JP  NZ  9000H ←条件つきジャンプ命令
JP  Z  9000H  ←
```

条件つきジャンプ命令のまえで演算をする

ゼロフラグやキャリーフラグが０になったり、１になったりするのは、まえに説明したように演算の結果によりますから、条件つきジャンプ命令を使うときは、条件つきジャンプ命令のまえで演算を行わなければなりません。

たとえば、196頁で説明したウエイトループで、ＪＲ（ジャンプ・リラティブ）命令がありましたが、ウエイトループのＪＲ命令も、

```
JR  NZ  E02BH
```

ＪＲ命令のあとにＮＺがありますから、条件つきジャンプ命令です。したがって、ゼロフラグが０のとき条件が成立してＥ０２Ｄ番地にジャンプします。この場合、つぎに示すように、ＪＲ　ＮＺ　Ｅ０２ＢＨ命令のまえで、ＤＥＣ　Ａで、Ａレジスタから１を引くという演算を行っています。

```
LD  A, FFH
    ∫
DEC  A ←条件つきジャンプ命令のまえで演算を行う
JR  NZ  E02BH
```

このように条件つきジャンプ命令の場合は、その命令のまえで演算を行わなければなりません。

スタックとスタックポインタ

スタックとは

144頁でレジスタについて説明しました。マシン語は、つぎのように、このレジスタにデータを記憶させて処理します。

```
LD  A, 30H ←――Aレジスタに30Hを入れる
LD  D, 40H ←――Dレジスタに40Hを入れる
ADD  A, D ←――Aレジスタの値とDレジスタの値を足す
```

これに対して、MSXベーシックでは、変数にデータを記憶させて処理します。

```
A=48 ←――変数Aに48を入れる
D=64 ←――変数Dに64を入れる
G=A+D ←――変数Aの値と変数Dの値を足して、変数Gに入れる
```

ところで、MSXベーシックの場合の変数は、データを記憶させるのに必要な数だけ作ることができますが、レジスタの数は、

A　B　C　D　E　H　L

の、たった7個です。レジスタの数がわずか7個でも、どんどんデータを記憶させていくことができればよいのですが、レジスタの場合も新たにデータを記憶させると、まえに記憶させたデータが消えてしまいます。とすると、わずか7個のレジスタでは、データを記憶させて処理するには足りないのではないかと、だれもが思います。もちろん、メモリに記憶させればよいと思う人もいるかもしれません。

たとえば、Bレジスタに30Hというデータが入れられているとします。つぎに、Bレジスタに20Hを入れて処理するというとき、ま

えに入れておいた３０ＨをＥ０５６番地のメモリに記憶させて、Ｂレジスタに２０Ｈを入れて処理するということです。

たとえば、つぎのようにです。

LD A, B←メモリに記憶させるBレジスタの30Hを入れる

LD (E056H), A←Aレジスタの30HをE056番地に入れる

このようにした場合に問題となるのは、Ｂレジスタの３０ＨをＥ０５６番地のメモリに入れたということを、私たちがおぼえておかなければならないということです。ひとつ、ふたつのデータなら、データを記憶させたメモリ番地をおぼえておいて、必要なときにレジスタに取りだしてくることもできますが、データの数が多くなるとおぼえきれなくなります。

そこで、わずか７個のレジスタでも、十分データの処理ができるように、**スタック**というものがあるのです。

スタックは、メモリの使い方のひとつの形

スタックは、データを記憶させておくメモリの使い方のひとつの形です。ふつう、レジスタのデータをメモリに記憶させるときは、データを記憶させるメモリの番地を指定しなければなりません。おなじように、メモリに記憶させたデータをレジスタに取りだすときも、データを記憶しているメモリの番地を指定しなければなりません。

LD (E056H), HL←HLレジスタのデータをE059番地に入れる

LD HL, (E980H)←E980番地のデータをHLレジスタに取りだす

ところが、スタックというメモリの使い方をすると、メモリの番地を指定する必要がなくなります。

ただし、スタックを使うには、スタックにデータを記憶させるＰＵＳＨ命令と、スタックからデータを取りだしてくるＰＯＰ命令がありますから、その命令をつかわなければなりません。

PUSH HL←HLレジスタのデータをスタックに入れる

POP HL←スタックに入れたデータをHLレジスタに取りだす

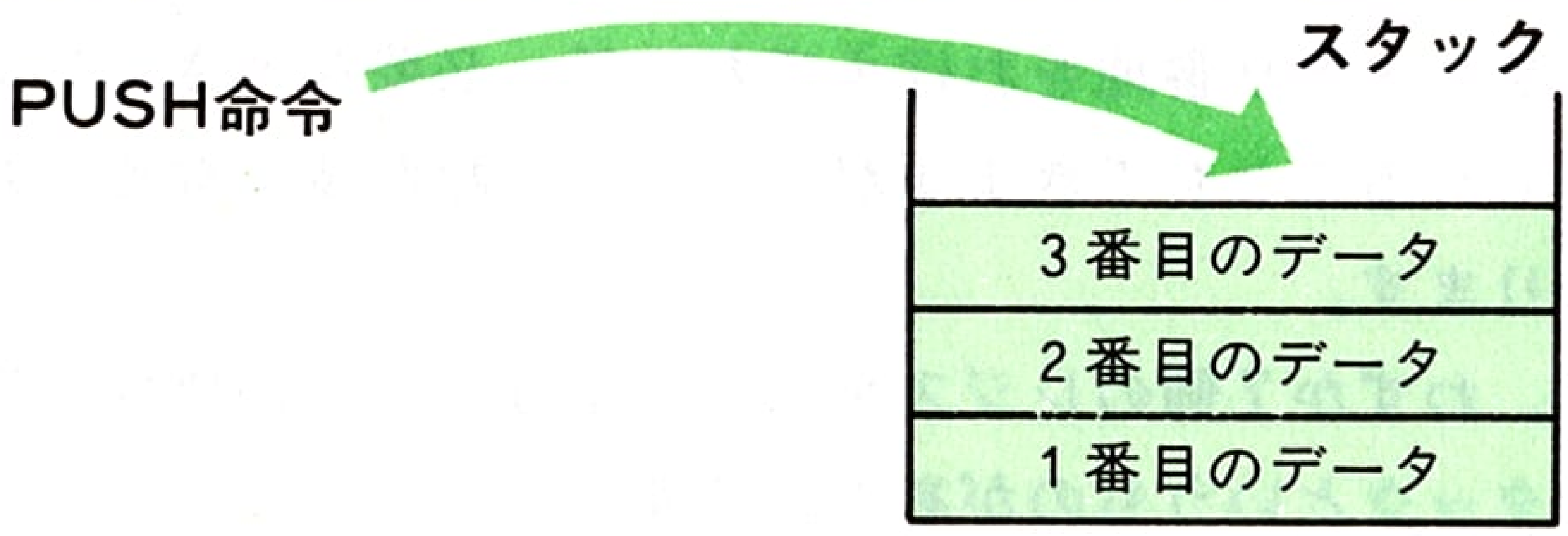
ＰＵＳＨ命令は、スタック退避(たいひ)命令ともいいます。ＰＯＰ命令は、

レジスタ復帰命令ともいいます。

ＰＵＳＨ命令＝スタック退避命令
ＰＯＰ命令＝レジスタ復帰命令

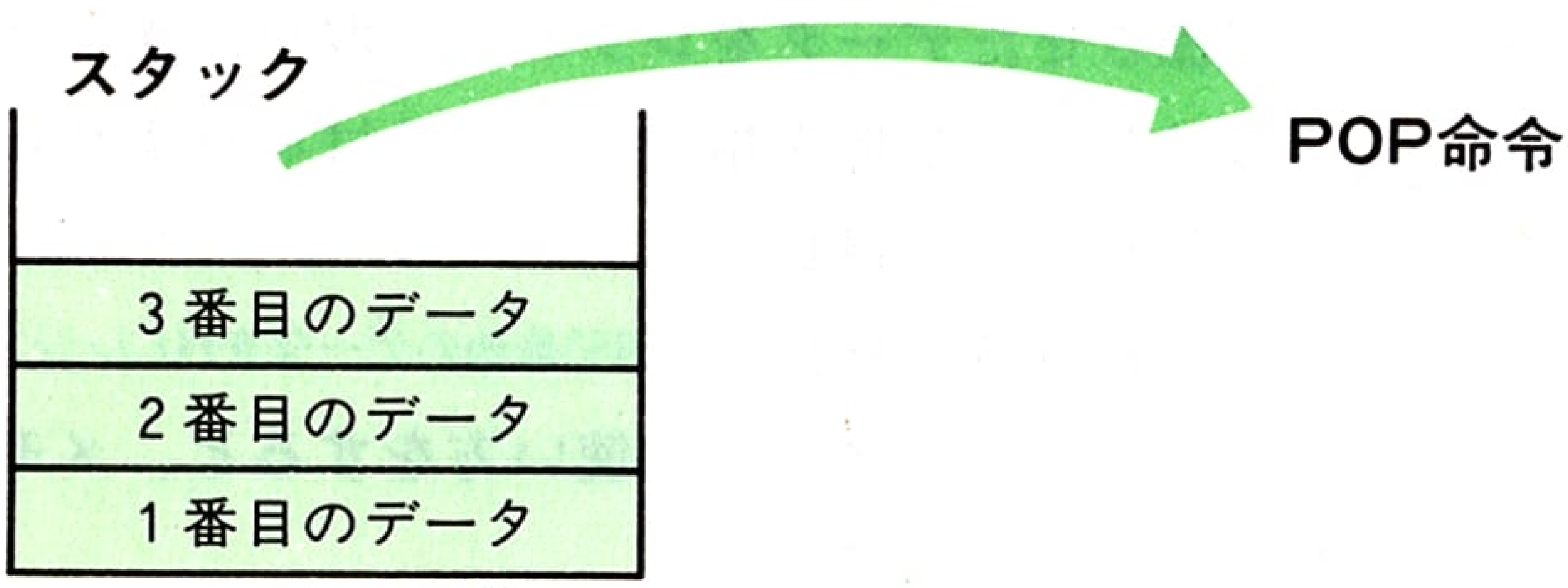
スタックへのデータの記憶のされ方

ＰＵＳＨ命令を使って、スタックにデータを記憶させると、つぎの図に示すように、最初のデータから順番に記憶されていきます。



したがって、最初にスタックに入れたデータが下になり、あとからスタックに入れたデータが上に積みかさねられて記憶されていきます。

ＰＯＰ命令で、スタックからデータをレジスタに取りだすときは、下になっているデータをさきに取りだすことはできませんから、上にあるデータから順番に取りだされてくることになります。



このように、あとから入れたデータをさきに取りだすことを、Last In First Out（ラスト・イン・ファースト・アウト）といいます。略してＬＩＦＯ（ライフォ）ということもあります。

ところで、実際にこのようなメモリがあるわけではありません。見かけのうえでの働きです。なぜならスタックは、私たちがマシン語のプログラムを記憶させるＲＡＭ領域を使っているからです。ＲＡＭ領

域のあるメモリ番地から、スタックとして決めて使っているにすぎないのです。

スタックをＲＡＭ領域の何番地のメモリからにするかは、このあと説明するスタックポインタで指定することができますが、スタックポインタでスタックのメモリ番地を指定するには多少の危険がともなうので、ふつうの場合は、ベーシック上のスタックを使います。ベーシック上のスタックを使うときは、自動的にスタックとして使う最初のメモリ番地がスタックポインタにセットされますから、めんどうな指定をすることなく、スタックを使うことができます。

なお、詳しいスタックの働きについては、スタックポインタと一緒でないと説明できないので、このあとのスタックポインタのところを参照してください。

PUSH命令を使うには

さて、レジスタのデータをスタックに記憶させるＰＵＳＨ（プッシュ）命令の使い方です。ＰＵＳＨ命令のあとには、レジスタペアをおきます。

PUSH 〈レジスタペア〉

レジスタペアとは、

BC　　DE　　HL

というように、８ビットのレジスタをふたつ組み合せたものです。レジスタペアにすると、16ビットのレジスタとなって、16ビットのデータを記憶することになりますが、スタックにデータを退避させるときは、たとえ８ビットのデータであっても、レジスタペアにしなければならないことに決まっています。

たとえば、Ｂレジスタに記憶されている８ビットのデータをスタックに退避させるときは、

PUSH　BC ←―― Bレジスタの8ビットのデータをスタックに退避

ということになります。Ｃレジスタに記憶されている８ビットのデ

ータをスタックに退避させるときも、おなじ形になります。

```
PUSH  BC
```

16ビットのデータをスタックに退避させるときは、そのままです。たとえば、ＤＥレジスタの16ビットのデータをスタックに退避させるときは、つぎのようになります。

```
PUSH  DE
```

POP命令を使うには

つぎは、スタックに退避させたデータを、レジスタに取りだしてくるＰＯＰ（ポップ）命令の使い方です。ＰＯＰ命令のあとにも、ＰＵＳＨ命令とおなじように、レジスタペアをおきます。

POP〈レジスタペア〉

ただし、ＰＯＰ命令におくレジスタペアは、ＰＵＳＨ命令でスタックにデータを退避させたときと、おなじレジスタペアになります。

たとえば、Ｂレジスタに記憶されている８ビットのデータをスタックに退避させるときは、ＰＵＳＨ命令にＢＣレジスタペアをおきます。そのデータをスタックからＢレジスタに取りだしてくるときは、ＰＯＰ命令にもＢＣレジスタをおきます。

```
PUSH  BC          POP  BC
```

レジスタペアはおなじにする

このようにすると、スタックに退避させたＢレジスタのデータが、Ｂレジスタに取りだされてきますが、これをつぎのようにすると、

```
PUSH  BC          POP  DE
```

レジスタペアが違う

Ｂレジスタのデータは、Ｄレジスタに取りだされることになりますから、ＰＵＳＨ命令でスタックに退避させたデータが、どのレジスタのデータであるかに注意して、もとのレジスタに正しくデータを戻すようにしなければなりません。

また、たとえばＤＥレジスタに記憶されている16ビットのデータを

スタックに退避させるときは、ＰＵＳＨ命令にＤＥレジスタペアをおきますから、そのデータをＰＯＰ命令で取りだしてくるときは、ＰＯＰ命令にもＤＥレジスタペアをおきます。

```
PUSH  DE          POP  DE
```

レジスタペアはおなじにする

このようにすると、スタックに退避させたＤＥレジスタペアの16ビットのデータが、ＤＥレジスタペアに取りだされてきます。

スタックポインタとスタック

さきに説明しましたが、ふつうの場合、スタックはベーシック上のスタックを使うので、単に、

```
PUSH  BC
POP  BC
```

という命令だけで、レジスタに記憶されているデータをスタックに退避させたり、また、スタックに退避させたデータをレジスタに取りだしたり（復帰）しています。

マシン語でベーシック上のスタックを使うのは、スタックポインタで、スタックとする最初のメモリ番地を指定したりするわずらわしさをさけて、簡単にスタックを使うことを目的としているためですが、スタックを詳しく知るためには、スタックポインタの働きとの関連でとらえなければなりません。

スタックポインタは、その頭文字を取ってＳＰとも書き表わします。

ＳＰは、スタックのメモリ番地を指示するレジスタ

ＳＰは、ＣＰＵのなかにある16ビットのレジスタで、スタックの入口にあたるメモリが、いま何番地であるかを指示する働きをします。ＳＰが16ビットのレジスタであるのは、メモリ番地が16ビット（16進数４桁）だからです。

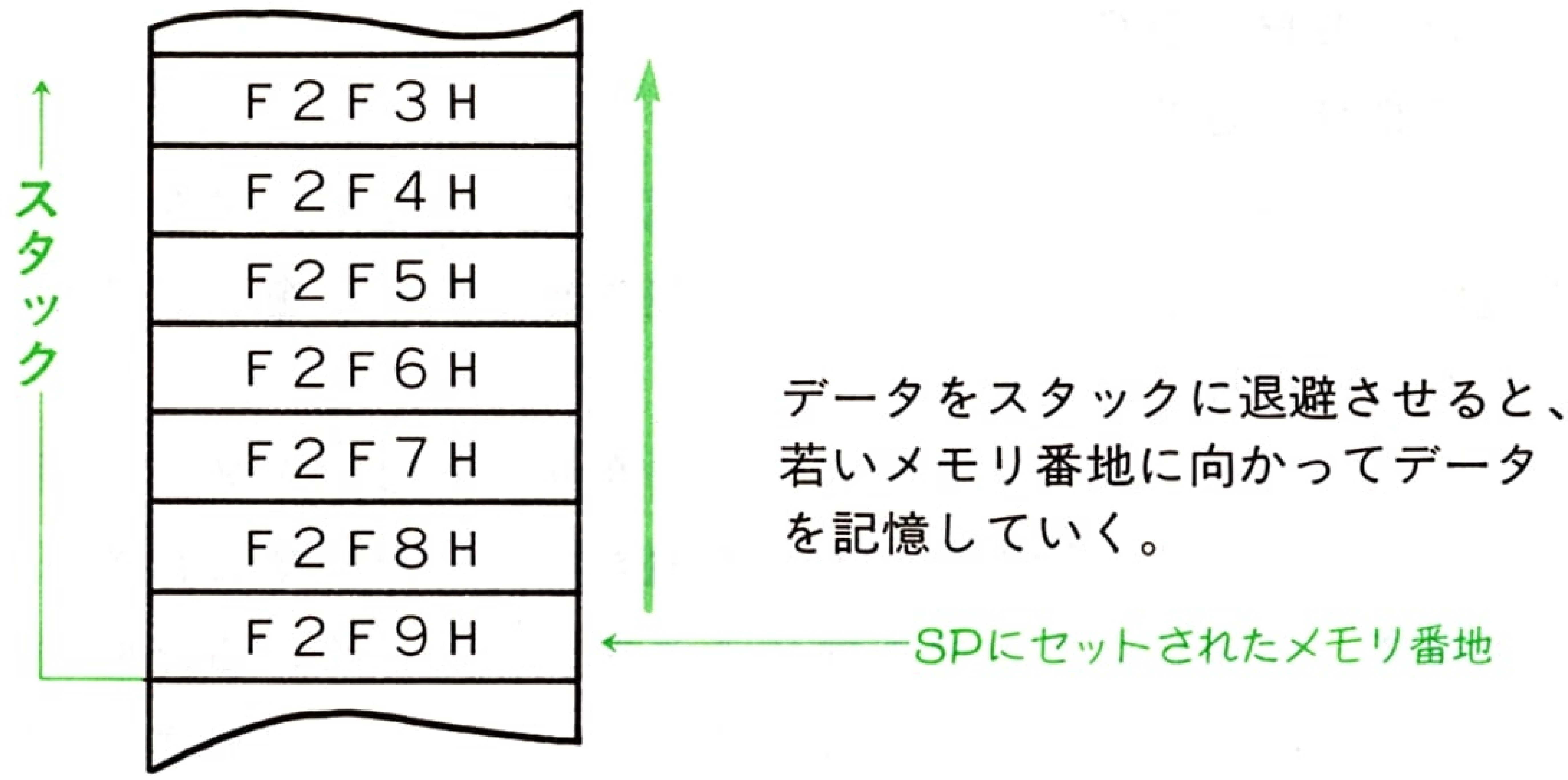
さて、マシン語モニタを実行させると同時に、ＳＰには、ベーシック上のスタックの最初のメモリ番地がセットされます。

SP←XX←ベーシック上のスタックの最初のメモリ番地がセットされる

ＭＳＸパソコンでは、何番地からスタックとなるのかわかりません。そこで説明の都合上、スタックの最初のメモリ番地を仮にＦ２Ｆ９Ｈとすることにします。このようにすると、マシン語モニタを実行させると同時に、ＳＰにはＦ２Ｆ９Ｈがセットされることになります。

SP←F2F9H

このようにＳＰにＦ２Ｆ９Ｈがセットされると、つぎの図に示すように、Ｆ２Ｆ９番地からスタックとなります。そして、スタックにデータを退避させると、Ｆ２Ｆ９番地より若い番地に向かって、データが記憶されていきます。



では、ＰＵＳＨ命令でデータをスタックに退避させてみます。まずＣレジスタに、８ビットのデータＦ３Ｈが記憶されているとします。

Cレジスタ＝F3H

８ビットのデータをスタックに退避させるときでも、ＰＵＳＨ命令にはレジスタペアをおきますから、

PUSH　BC

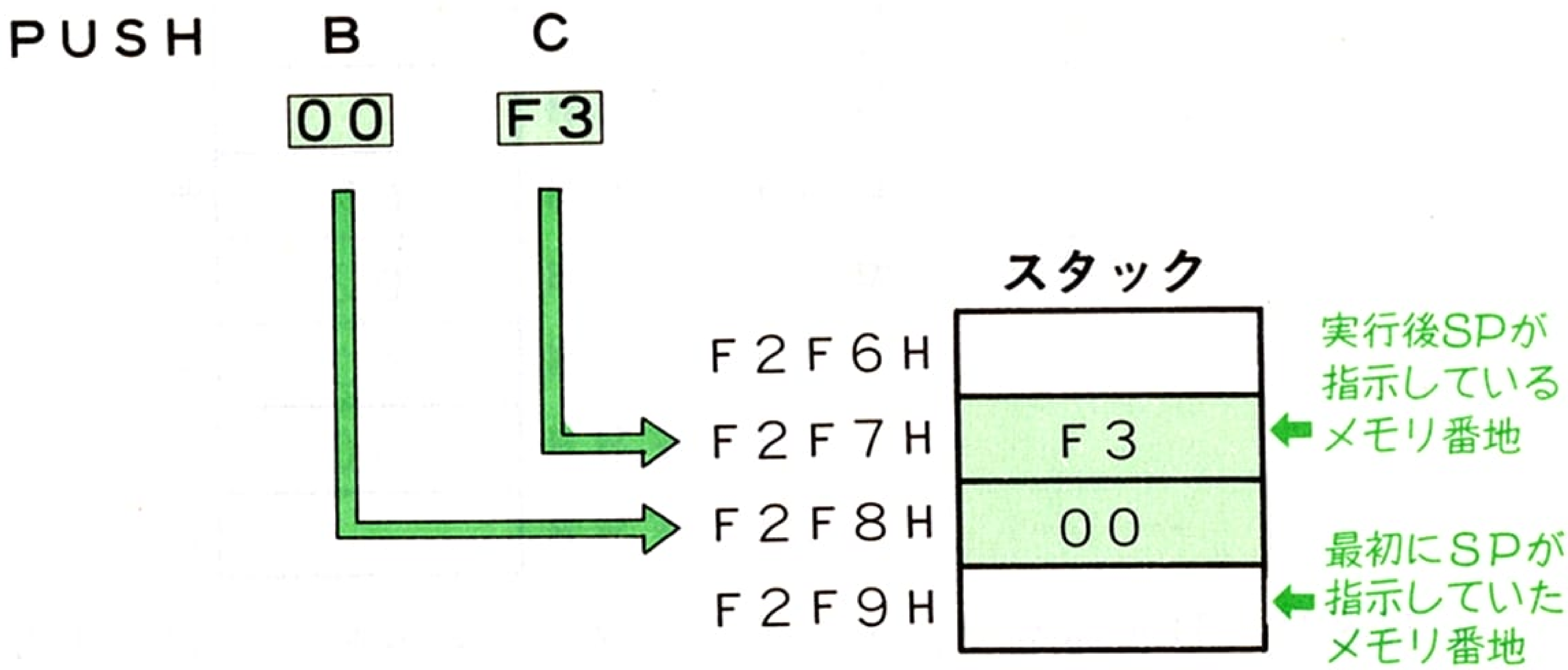
となります。この場合、Ｂレジスタは００Ｈ、Ｃレジスタは記憶されているデータのＦ３Ｈとなります。

PUSH　　B　　C

00　F3

ＰＵＳＨ　ＢＣ命令が実行されると、つぎの図に示すようにＢレジスタの００Ｈが、ＳＰにセットされて指示しているＦ２Ｆ９番地よりひとつ若い、Ｆ２Ｆ８番地のメモリに退避させられて記憶されます。

またＣレジスタのＦ３Ｈは、それよりもひとつ若いＦ２Ｆ７番地のメモリに退避させられて記憶されます。



このようにＰＵＳＨ　ＢＣ命令で、データをスタックに退避して記憶すると、ＳＰは、それまで指示していたＦ２Ｆ９番地からマイナス２したＦ２Ｆ７番地を指示します。

では、つづけてＤＥレジスタペアに記憶されているデータを、スタックに退避させることにします。ＤＥレジスタペアには、Ｄ２Ｅ５Ｈが記憶されているとします。

DEレジスタペア＝D2E5H

このように、ＤＥレジスタペアにＤ２Ｅ５Ｈが記憶されている場合、Ｄ２Ｅ５Ｈというデータが、ＤＥレジスタペアにどのように記憶されているかというと、ＤレジスタにＤ２Ｈ、ＥレジスタにＥ５Ｈが記憶されています。

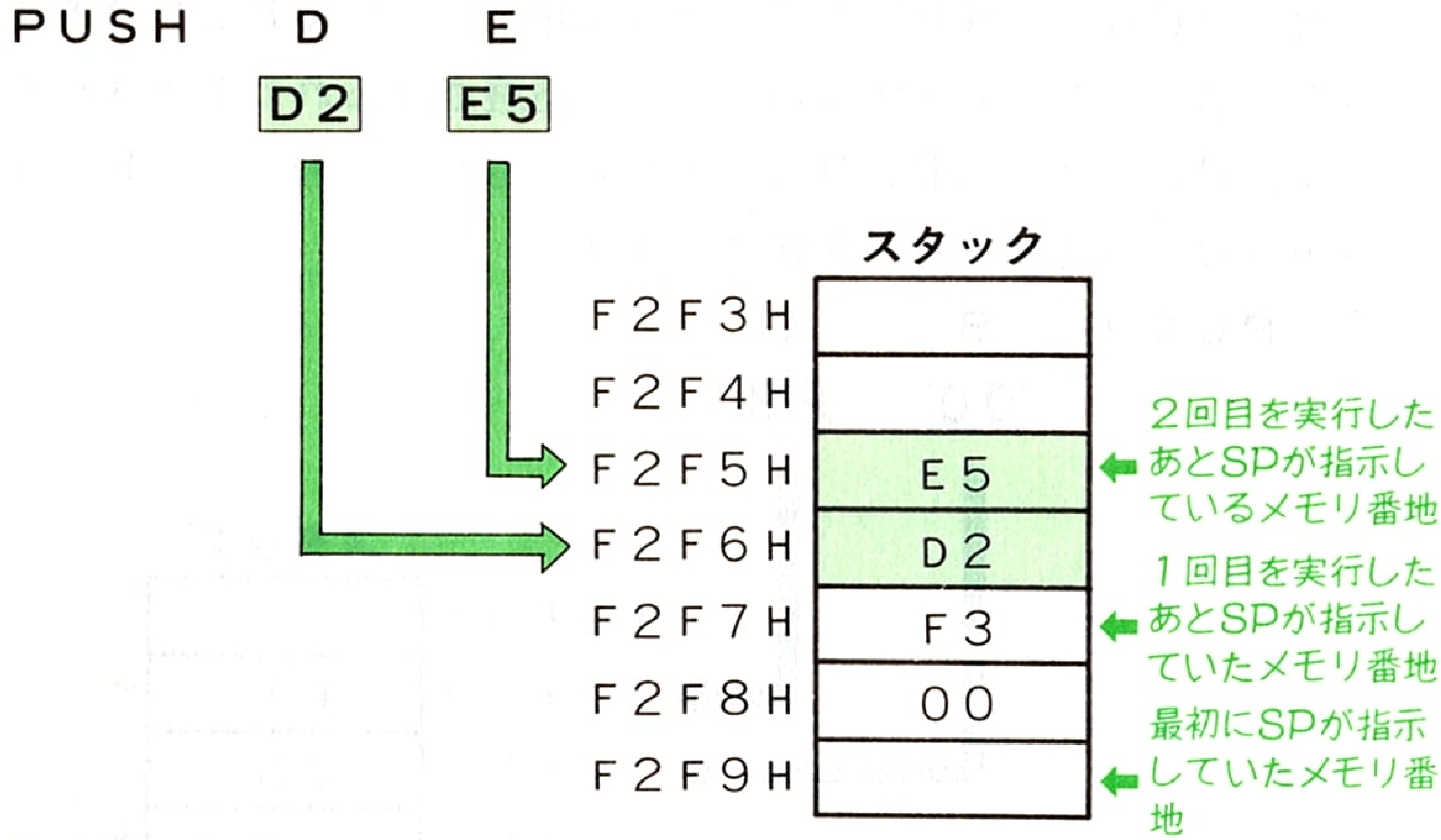
PUSH　D　E

D2　E5

したがって、ＰＵＳＨ　ＤＥ命令が実行されると、つぎの図に示すように、ＤレジスタのＤ２Ｈが、ＳＰが支持しているメモリ番地よりひとつ若いＦ２Ｆ６番地のメモリに退避させられて記憶されます。

また、ＥレジスタのＥ５Ｈは、それよりもうひとつ若いＦ２Ｆ５番

地のメモリに退避させられて記憶されます。



ＰＵＳＨ　ＤＥ命令で、データをスタックに退避して記憶すると、ＳＰは、１回目のＰＵＳＨ　ＢＣ命令を実行したあと指示しているＦ２Ｆ７番地から、マイナス２したＦ２Ｆ５番地を指示します。

このようにＳＰは、ＰＵＳＨ命令が実行されると、それまで指示していたメモリ番地からマイナス２したメモリ番地を指示しますから、ＰＵＳＨ命令を続けて行った場合、退避されるデータは若いメモリ番地に向かって、つぎつぎに記憶されていくことになります。

POP命令で取りだすときは

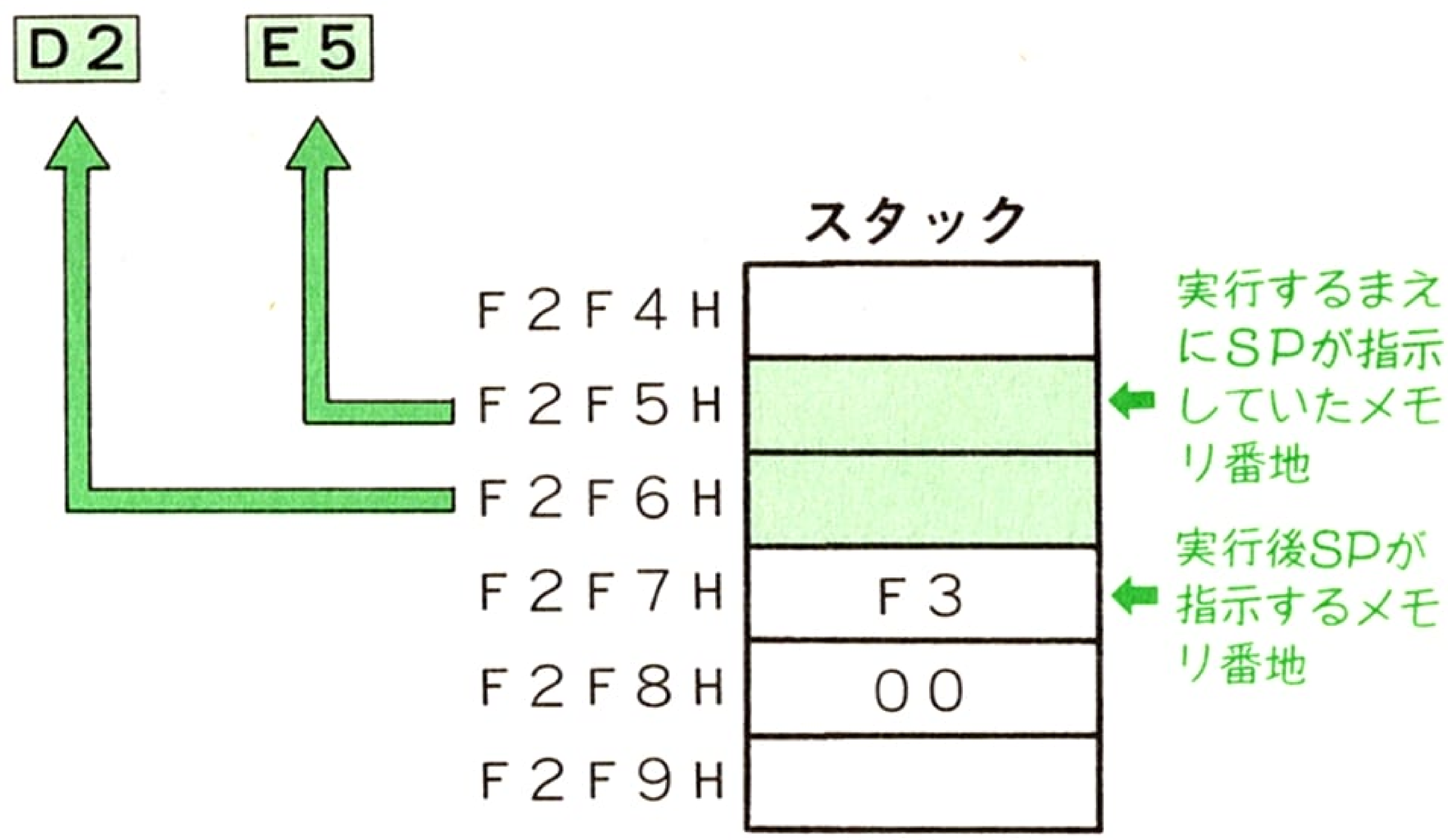
ではつぎは、ＰＵＳＨ命令でスタックに退避させたＢＣレジスタのデータ、ＤＥレジスタのデータを、ＰＯＰ命令でＢＣレジスタと、ＤＥレジスタに取りだす（復帰）ことにします。

取りだすデータは、ＳＰが指示しているメモリ番地からとなります。したがって、最初にスタックに退避させたＢＣレジスタのデータのうえに、あとから退避させたＤＥレジスタのデータがあることになりますから、あとから退避させたＤＥレジスタのデータを取りだして、そのあとに、最初に退避させたＢＣレジスタのデータを取りだすことになります。

POP　DE

この命令を実行すると、スタックのF２F５番地とF２F６番地に退避させたDEレジスタのデータが、DEレジスタに取りだされます。

POP　D　E



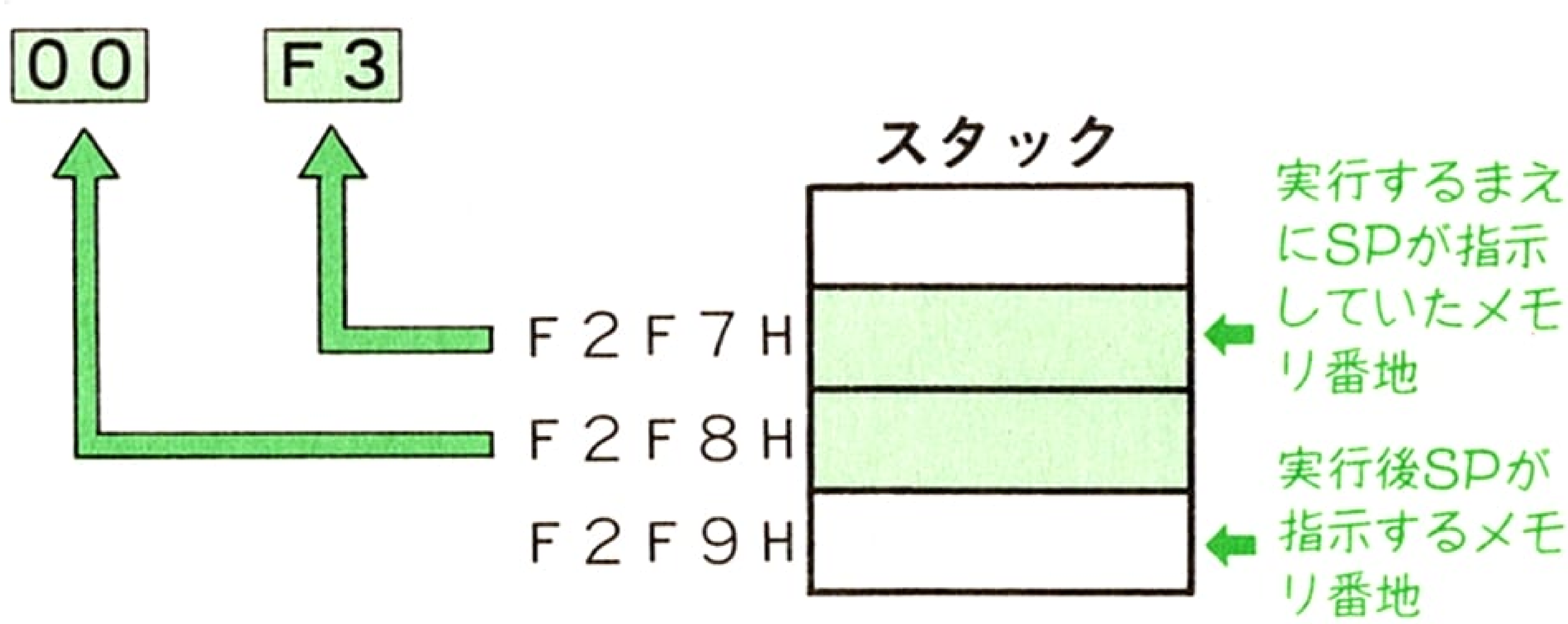
データがDEレジスタに取りだされると、SPは、それまで指示していたF２F５番地に2をプラスして、F２F７番地を指示します。

つぎに、

POP　BC

を実行すると、スタックのF２F７番地とF２F８番地に退避させたBCレジスタのデータが、BCレジスタに取りだされます。

POP　B　C



BCレジスタにデータが取りだされると、SPは、それまで指示していたF２F７番地にプラス2して、最初にSPが指示していたF２F９番地を指示します。

これが、スタックポインタの働きとスタックの関係です。

イラスト　波木博信

MSX
わかるマシン語入門

1986年12月25日　発　行　©

著　　者　田　中　一　郎
　　　　　大　沢　昭　二
発 行 者　富　永　弘　一
印 刷 所　慶昌堂印刷株式会社

発行所　東京都台東区台東２丁目24　株式会社　新星出版社
電話(831)0743　郵便番号110　振替東京4-72233

ISBN4-405-06065-7

ゲームで覚える
MSXベーシック
川中篤胤著

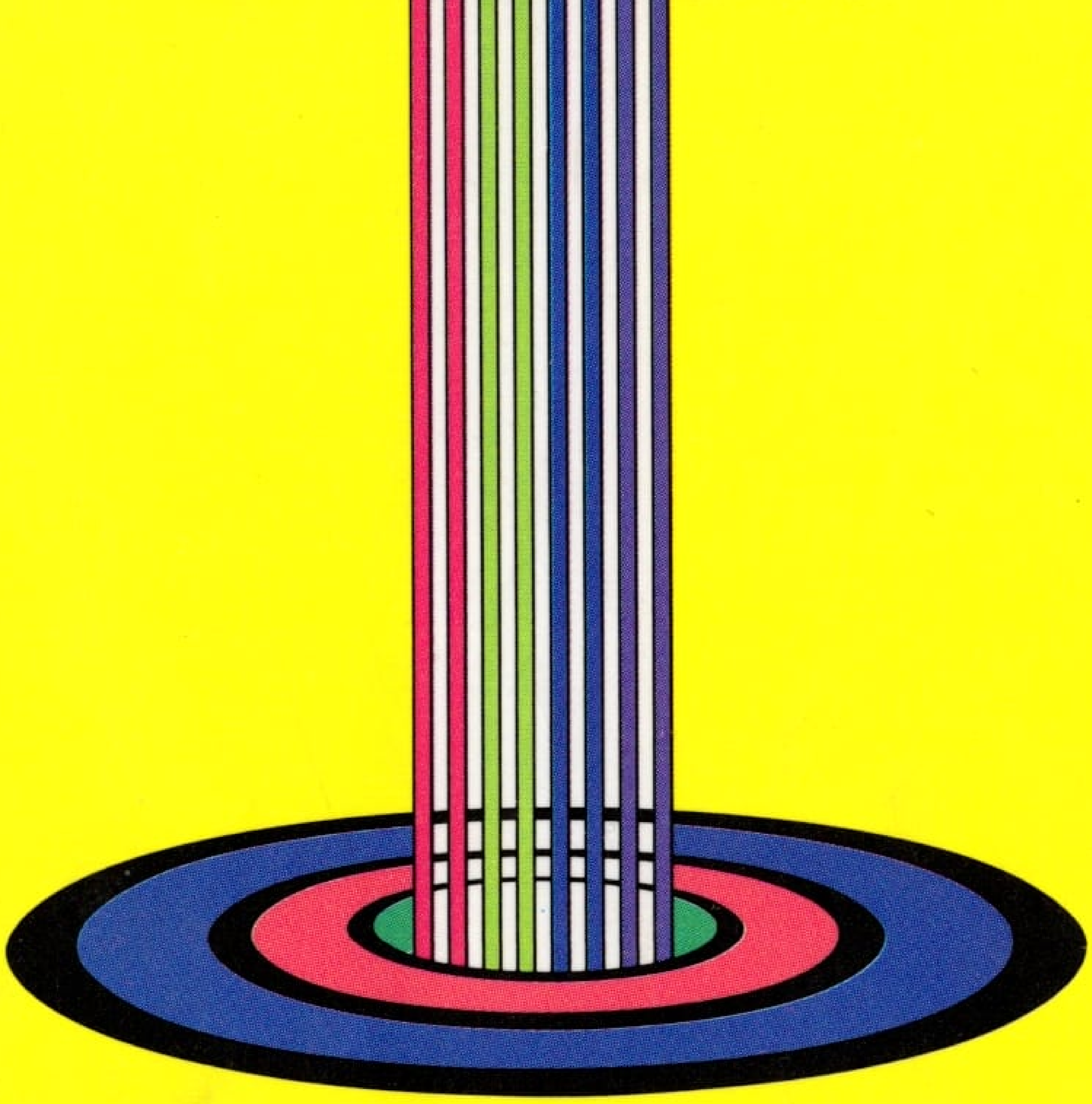
マイコン用語辞典
関口 泰・山科敦之著

MSX
ベーシック用語辞典
田中一郎・小山郁夫著

ベーシック決まり文句
大橋 均・田中一郎著

わかるMSX
マシン語入門

マシン語の一般知識とプログラミング



新星出版社

定価1200円

ISBN4-405-06065-7 C2055 ¥1200E